

Cooperative Defense Against Pollution Attacks in Network Coding Using SpaceMac

Anh Le, Athina Markopoulou
University of California, Irvine

Abstract—Intra-session network coding is known to be vulnerable to pollution attacks. In this work, first, we introduce a novel homomorphic MAC scheme called SpaceMac, which allows an intermediate node to verify if its received packets belong to a specific subspace, even if the subspace is expanding over time. Then, we use SpaceMac as a building block to design a cooperative scheme that provides complete defense against pollution attacks: (i) it can detect polluted packets early at intermediate nodes and (ii) it can identify the exact location of all, even colluding, attackers, thus making it possible to eliminate them. Our scheme is cooperative: parents and children of any node cooperate to detect any corrupted packets sent by the node, and nodes in the network cooperate with a central controller to identify the exact location of all attackers. We implement SpaceMac in both C/C++ and Java as a library, and we make the library available online. Our evaluation on both a PC and an Android device shows that (i) SpaceMac's algorithms can be computed quickly ($\sim 28 \mu\text{s}$ in C/C++) and efficiently ($\sim 64 \text{ KB}$ in C/C++), and (ii) our cooperative defense scheme has both low computation ($\sim 190 \mu\text{s}$ in C/C++) and low communication ($\sim 2\%$) overhead, significantly less than other comparable state-of-the-art schemes.

Index Terms—Byzantine attacks, pollution attacks, network coding, attack detection, attack location, homomorphic MAC

I. INTRODUCTION

THE network coding paradigm advocates that intermediate nodes in a network should mix incoming packets instead of simply forwarding them, and receivers should decode to obtain the original packets. This idea, originally introduced by Ahlswede *et al.* [1], has been shown to bring benefits in terms of throughput and distributed operation of networks, and has received much attention. In this work, we consider networks that employ intra-session linear network coding.

An inherent weakness of network coding is that it is particularly vulnerable to pollution (a.k.a. Byzantine) attacks. Malicious nodes can inject corrupted packets into a network. These packets are combined and forwarded by downstream nodes, causing a large number of corrupted packets propagate in the network. This wastes resources of the network, such as bandwidth and CPU time, and eventually prevents the decoding of the original packets at the receivers. The detrimental effect of pollution attacks has been shown through both theoretical analysis [2] as well as experimentation [3], [4].

Proposed defense mechanisms against pollution attacks can be classified into three categories: error correction [5]–[8], attack detection [4], [9]–[18], and locating attackers [19], [20].

In this paper, we are interested in the latter two approaches. In particular, we set out to design a complete defense system that can not only detect the pollution attack in a timely manner but also accurately locate and eliminate all pollution attackers. This allows for dealing with any attack early and at its root. To the best of our knowledge, none of the existing defense mechanisms can provide this level of protection.

To this end, we first propose a novel homomorphic message authentication code (MAC) scheme for expanding spaces called SpaceMac. SpaceMac allows a node to verify if its received packets belong to a specific subspace, even if the subspace is expanding over time. We then design our novel cooperative defense system which includes both a detection scheme and a locating scheme, using SpaceMac as their building block. Our detection scheme relies on SpaceMac to force intermediate nodes to send only linear combinations of packets that they actually receive from their parents. Parents and children of any intermediate node cooperate to detect corrupted packets sent by the intermediate node. Our locating scheme uses SpaceMac to force nodes in the network to truthfully cooperate with a central controller so that the controller can exactly locate the pollution attackers. Finally, by leveraging multiple generations, our scheme is able to deal with a large number of colluding attackers.

The main contribution of this paper is twofold:

- **The design and implementation of SpaceMac:** We describe the construction of SpaceMac and provide a formal security proof for the construction. We implement SpaceMac in both C/C++ and Java as a ready-to-use library. Our Java implementation is compatible with the current Android OS (Android 2.2 Froyo). We make the library available online [21].
- **The design of a novel cooperative defense system based on SpaceMac:** To the best of our knowledge, our defense system is the first that meets all of the following requirements simultaneously: (i) it can provide timely in-network detection, (ii) it can exactly locate all pollution attackers, (iii) it can deal with a large number of colluding attackers, and (iv) it has low communication and computation overhead.

We have extensively evaluated the computation overhead of SpaceMac's algorithms and both the computation and communication overhead of our defense scheme through real implementation in both C/C++ and Java, and on both a PC and an Android device (Samsung Captivate). Our evaluation results show that all three algorithms of SpaceMac (Mac, Combine,

and Verify) can be computed efficiently (requiring 64 KB of memory in C/C++ or 128 KB in Java) and also quickly on a PC ($< 28 \mu\text{s}$ in C/C++) and even on a smart phone ($< 2.3 \text{ ms}$). Evaluation results also demonstrate that when implementing our defense scheme, nodes in the network introduce very small computational delay (in the order of sub-millisecond on the PC and millisecond on the smart phone). Moreover, our defense scheme was shown to introduce very low communication overhead (2%), significantly less than other comparable state-of-the-art schemes. Lastly, through a simulation in Python, we show that in a medium-size network of 50 nodes, our locating scheme can quickly locate all, even colluding attackers (20 attackers in about 1 second).

The rest of this paper is organized as follows. Section II discusses existing approaches to protect network coding against pollution attacks. Section III formulates the problem, describes the threat model, and discusses our design goals. Section IV presents our key observations and the overview of our approach. Section V presents the construction of SpaceMac and the formal security proof. Section VI describes our detection scheme. Section VII describes our locating scheme. Section VIII analyzes the security of our proposed schemes. Section IX presents our implementation and the evaluation results. Finally, we conclude in Section X.

II. RELATED WORK

There are three main approaches in the literature to combat pollution attacks: error correction, attack detection, and locating attacks. Below, we discuss each one in detail.

A. Error Correction

One of the earliest work on error correction for network coding is by Cai and Yeung [5]. The study in [5] introduces network error-correcting codes as a generalization of the traditional error correction codes. In a related study by Zhang [8], the minimum rank of a network error correction code is defined; this concept is analogous to the minimum distance in classical coding theory. Based on this concept, a network error correction codes similar to an ordinary linear network single source multicast code is designed. Jaggi *et al.* [22] consider packets from an attacker as an additional source and add redundancy at the source so that the receivers can decode both sources: the original source and adversary's source. In [7], Koetter and Kschischang proposed a coding metric on subspaces and a minimum distance decoder, which give rise to codes capable of correcting certain combinations of errors and erasures.

These information theoretic approaches, which aim at correcting errors at the receivers, offer only limited security against restricted types of adversaries. These approaches assume that the adversaries can only corrupt a small number of edges and packets. Also, the amount of redundancy, which can also be considered as the communication overhead, typically increases proportional to the number of corrupted packets or adversaries. Furthermore, these approaches do not detect and drop corrupted packets, and thus are unable to prevent the

corrupted packets from propagating in the network and using up resources.

In contrast, our defense scheme is able to provide timely detection of the attack, thereby allowing for early filtering of corrupted packets. More importantly, our approach can accurately locate the attackers to eliminate them from the network. Furthermore, our approach can deal with more powerful adversaries, *i.e.*, adversaries who pollute arbitrary number of packets and even colluding adversaries. However, we make an assumption on the adversaries' computational power, which is typical of all approaches utilizing cryptographic primitives.

B. Attack Detection

We first describe approaches that do not use homomorphic cryptography. In [15], Ho *et al.* show that randomized network coding can be extended to provide end-to-end attack detection, *i.e.*, allow the receivers to detect any corrupted packet. The extension requires the source node to include in each source packet some additional hash blocks calculated from the source data blocks using polynomial functions. More recently, Kehdi and Li [16] propose an in-network detection scheme which exploits subspace properties of network coding. In their scheme, intermediate nodes verify the integrity of a vector by checking if it belongs to the subspace spanned by the source vectors. Null keys, which are vectors orthogonal to all the combinations of the source vectors, are used for the verification. This scheme is not collusion resistant: multiple nodes can collude to infer the null keys and make benign nodes accept polluted vectors. Yu *et al.* [17] use simple XOR checksums and exploit probabilistic key pre-distribution to provide in-network detection. This scheme, however, has significant communication overhead due to the aggregation of authentication tags; moreover, the scheme is c -collusion resistant for some pre-determined constant c , *i.e.*, the scheme becomes vulnerable when there are more than c colluding attackers. In [3], Dong *et al.* design a linear transformation checksums to be used with a time-based authentication scheme to provide in-network detection. This scheme requires time synchronization among nodes in the network and frequent public key verification (one per generation).

A significant number of homomorphic cryptographic primitives ranging from hashes, signatures, to MACs, has been designed specifically to combat pollution attacks in network coding. In [9], Krohn *et al.* proposed a homomorphic hash scheme for verification of rateless erasure codes. Gkantsidis and Rodriguez [4] later propose probabilistic checking and cooperation mechanisms among nodes to reduce the computation overhead when using Krohn *et al.*'s scheme in peer-to-peer file distribution systems. Li *et al.* [23] also propose a hash-based scheme based on a trapdoor one-way permutation which can avoid the pre-distribution of the hash blocks. In [14], Zhao *et al.* propose a signature scheme where the source derives authentication information from a vector orthogonal to the source space. In [13], Charles *et al.* propose a signature scheme based on aggregate signatures. Recently, Boneh *et al.* presents a signature scheme built on bilinear maps [12]. All of the hash-based and signature-based approaches suffer from

a common drawback: they require expensive computation at intermediate nodes, either modular exponentiation or bilinear map, which results in high latency.

Recently, two homomorphic MAC schemes are proposed by two groups of researchers: Agrawal and Boneh [10], and Li *et al.* [11]. The scheme in [10] relies on cover free set systems for pre-distributing keys to provide in-network detection, and thus, only c -collusion resistant. This scheme is also susceptible to tag-pollution attacks, where malicious nodes tamper with some subset of tags of a packet. We discuss about this type of attack in detail in Section VIII-C. The scheme in [11] is collusion resistant (rather than c -collusion resistant) as well as resistant against tag-pollution attacks; however, it requires time synchronization among nodes in the network. Both schemes have low computation overhead since they only require simple addition and multiplication operations at intermediate nodes for both combining MAC tags and tag verification. In a more recent work [18], Zhang *et al.* introduce both a homomorphic MAC and a homomorphic signature scheme and propose a hybrid approach that uses both. This approach is not susceptible to tag-pollution attack but only c -collusion resistant. This approach also has lower computation overhead than signature-based approaches; however, the overhead is still significantly higher than the other two MAC-based approaches due to the expensive exponentiation operations required by the signature scheme.

Our SpaceMac scheme is inspired by the MAC scheme in [10]; however, our scheme allows intermediate nodes to sign *-expanding* over time- subspaces. This stands in stark contrast to [10], which allows for signing only fixed subspaces. Our scheme can be considered as a generalization of the scheme in [10]. Detailed description and comparison are provided in Section V. SpaceMac was originally introduced for authenticating expanding subspaces and locating the attackers in our preliminary work [24]. In this paper, we show that the ability to authenticate expanding subspaces can also be utilized to provide timely in-network detection without requiring time synchronization as in [11].

C. Locating Attackers

Compared to the other two categories, locating attackers has received less attention. An early work by Jafarisiavoshani *et al.* [19] leverages the subspace properties of randomized network coding to locate pollution attackers. The main observation is that packets sent by a node have to belong to the space spanned by source packets and also the space spanned by the packets the node receives from its parents. Using this observation, in a general network topology having a single attacker, the authors can locate the attacker with an uncertainty of at most two nodes; when there are multiple attackers, the uncertainty is within a set of nodes including the attackers and their parents and children. Our scheme builds on and significantly improves this work: we make it possible to pinpoint the exact location of the attackers, even in the case where there are multiple colluding attackers, thereby allowing for the removal of all attackers.

Recently, Wang *et al.* [20] introduce a light-weight non-repudiation protocol ensuring that (i) a malicious node that

Symbol	Explanation
\mathbf{v}_i	source packet, formed by augmenting $\hat{\mathbf{v}}_i$ with coefficients
n	the number of symbols carried in each packet $\hat{\mathbf{v}}_i$
m	size of a generation
$\mathcal{P}_N, \mathcal{C}_N$	the parent set and the child set of N
$\mathcal{S}, \mathcal{R}, \mathcal{I}$	source node, receiver set, intermediate node set
Π^S	source space, spanned by source packets
Π^N	sending space, spanned by packets sent by N
Π_N^P, Π_N	received space of N from P and from all parents

TABLE I
Notation

injected a polluted packet cannot deny its behavior and (ii) a malicious node cannot disparage any innocent node. They build a defense scheme based on the protocol to identify malicious nodes. The scheme in [20] performs flooding of multiple checksums of all the packets sent by the source to all the nodes, which incurs significant communication overhead. Moreover, because the success of the locating scheme relies on the successful reception of the checksums at every node, this scheme is vulnerable to colluding attackers. Finally, this scheme is unable to locate all the attackers. We use their non-repudiation protocol as a building block of our locating scheme. However, unlike the scheme in [20], our scheme is able to locate all, even colluding attackers, without the need of checksums.

Finally, compared to our prior preliminary work [24], where we first presented SpaceMac and our locating scheme, this work is significantly improved and extended by the following three novel contributions: (i) we describe a novel construction of SpaceMac, whose algorithms are significantly more computationally efficient than our previous construction; (ii) we describe a novel detection scheme built on SpaceMac that can provide in-network detection with low overhead; and (iii) we implement the SpaceMac library in both C/C++ and Java and we extensively evaluate both SpaceMac and the proposed defense scheme on both a PC and an Android device.

III. PROBLEM FORMULATION

In this section, we describe the notation that we use to express network operations in a multicast session with intra-session coding. In addition, we describe the threat model and the design goals of our defense system.

A. Network Model and Operation

We follow the notation used in [10] and [19]. Consider a fixed, directed acyclic graph (DAG), denoted by $G = (\mathcal{V}, \mathcal{E})$. There is a single source node S that multicasts packets to a set of receivers, denoted by \mathcal{R} . Denote the set of intermediate nodes as \mathcal{I} , i.e., $\mathcal{I} = \mathcal{V} \setminus \{\mathcal{R} \cup \{S\}\}$. Nodes in \mathcal{I} perform generation-based linear network coding. A generation consists of m packets, $\hat{\mathbf{v}}_1, \dots, \hat{\mathbf{v}}_m$, in an n -dimensional linear space \mathbb{F}_q^n , where m, n and q are fixed ahead of time and $q \gg 1$. The source augments every packet $\hat{\mathbf{v}}_i$ with m additional symbols, which are the coefficients of $\hat{\mathbf{v}}_i$. The resulting packets, \mathbf{v}_i 's, called *source packets*, have the following form:

$$\mathbf{v}_i = (\hat{\mathbf{v}}_i, \underbrace{0, \dots, 0, 1, 0, \dots, 0}_m) \in \mathbb{F}_q^{n+m}.$$

Note that if a packet \mathbf{y} is a linear combination of the packets \mathbf{v}_i 's, then the last m symbols of packet \mathbf{y} contain the global linear combination coefficients. The source S sends the packets \mathbf{v}_i 's to the network. Denote the subspace spanned by packets \mathbf{v}_i 's by $\Pi^S \subseteq \mathbb{F}_q^{n+m}$. We refer to Π^S as the *source space*.

We use \mathcal{P}_N and \mathcal{C}_N to denote the sets of parents and children of a node N , respectively. Each intermediate node N receives from \mathcal{P}_N some packets, which are linear combinations of the source packets. It then creates linear combinations of the received packets and sends them to its adjacent downstream nodes. We use $\Pi_N^P(t) \subseteq \mathbb{F}_q^{n+m}$ to denote the space spanned by the packets received by node N from a parent node P , $P \in \mathcal{P}_N$, up to time t . We further use $\Pi_N(t)$ to denote the space spanned by all the packets received by node N from all its parents up to time t . We also denote the space spanned by all the packets sent by a node N up to time t as $\Pi^N(t)$. When there is no ambiguity, we omit the time index t . A receiver $R \in \mathcal{R}$ can successfully decode the original source packets using Gaussian elimination if its received space Π_R equals the source space Π^S .

If all the nodes in the network are benign, then the space spanned by packets sent by N , Π^N , must be a subspace of the space spanned by the packets that N receives, Π_N . This is a property of networks that implement random linear network coding. This observation was also made in [19]. Formally,

Lemma 1. *If every node in the network is benign then for every node N , $\Pi^N(t) \subseteq \Pi_N(t)$.*

Also, observe that for any parent P of N , both Π_N^P and Π_N expand over time. Formally, $\Pi_N^P(t_0) \subseteq \Pi_N^P(t_1)$ and $\Pi_N(t_0) \subseteq \Pi_N(t_1)$, for all $t_0 \leq t_1$.

Furthermore, when all the intermediate nodes $N \in \mathcal{I}$ are benign, the incoming spaces of all the intermediate nodes and the receivers are subspaces of the source space. Assume that there is a pollution attacker in the network. The attacker combines a subspace $\Pi^* \not\subseteq \Pi^S$ with its incoming space and sends the resulting packets to its children; as a result, the incoming spaces of these children are not subspaces of Π^S . Formally, for every node N , $N \in \mathcal{V} \setminus \{S\}$, its incoming space Π_N^P from its parent P can be decomposed as follows: $\Pi_N^P = \Pi_N^{S_P} \oplus \hat{\Pi}_N^P$, where \oplus denotes the direct sum of spaces, $\Pi_N^{S_P} \stackrel{\text{def}}{=} \Pi^S \cap \Pi_N^P$, and $\hat{\Pi}_N^P$ contains packets not belonging to Π^S . Table I summarizes the notation used in this paper.

Following the framework in [19], we define a *polluted directed edge* as follows:

Definition 1. *A directed edge is polluted if it transmits any packet which is not a linear combination of the source packets.*

The following lemma, adapted from [19], is a direct consequence of the definition:

Lemma 2. *A directed edge $e(P, N)$ is polluted if and only if $\Pi_N^P \not\subseteq \Pi^S$.*

B. Threat Model

We assume that both the source and the receivers are trustworthy but the intermediate nodes may be malicious. The

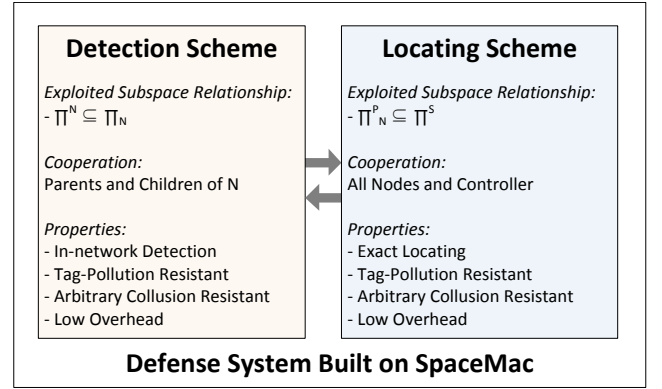


Fig. 1. Cooperative defense system's main components, their exploited subspace relationships, and their properties

case when the receivers are malicious is discussed separately in Section VIII-E. We assume that the network may have multiple pollution attackers. They may be located at an arbitrary set of intermediate nodes in the network. Each attacker may inject corrupted packets, *i.e.*, packets that do not belong to the source space, into a single or multiple downstream edges to pollute the network. They may also modify other data associated with the packets, such as, authentication tags. Successful modification of authentication tags constitutes an attack called “tag pollution,” which could be as devastating as a pollution attack [11]. Tag pollution attacks are discussed separately in Section VIII-C. We consider both cases where the pollution attackers launch their attacks independently or collude and coordinate their attacks. We further assume that the attackers are aware of our defense scheme, *i.e.*, the construction and application of SpaceMac; however, similar to other cryptographic approaches, we assume that the attackers’ running time is polynomial in the security parameter.

C. Design Goals

With this threat model in mind, we set out to design a defense system with the following goals:

- (1) *In-network Detection.* Any intermediate node in the network should be able to detect the attack as soon as its malicious parent injects a corrupted packet into the network. This prevents corrupted packets from polluting the downstream edges.
- (2) *Exact Locating.* The location of all pollution attackers should be precisely identified. This allows for the removal of the attackers from the network.
- (3) *Arbitrary Collusion Resistance.* The system should be able to cope with multiple pollution attackers when they attack independently as well as when they collude. In particular, the defense system should be able to remove the attackers from the network even when they collude.
- (4) *Low Overhead.* The defense system should have low computation and low communication overhead. In particular, the system should require a little amount of computing from the nodes in the network and should not introduce a large amount of traffic, *e.g.*, bandwidth of the MAC tags, to the network.

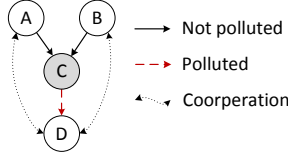


Fig. 2. An example illustrates how SpaceMac helps to detect pollution attacks. Using SpaceMac, A and B are able to sign the expanding space Π_C (the received space of C) and D is able to verify any packet sent by C to see if it belongs to Π_C . If there is a packet sent by C that is not in Π_C , the attack is detected by D . The cooperation among A , B , and D helps to detect the attack from C .

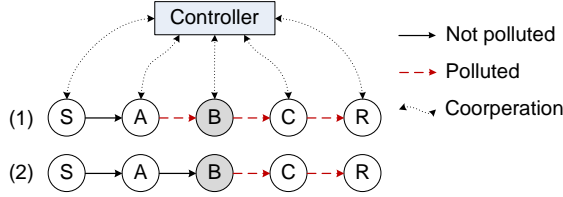


Fig. 3. An example of inferring an attacker's location using information about polluted edges from [19]: The attacker is at node B . Scenarios (1) and (2) correspond to the sets of polluted edges when the attacker lies and is honest about its incoming subspace, respectively. The controller can narrow down the attacker to two nodes: A and B , as they initiate polluted edges. The cooperation among nodes in the network and the controller helps the locating process.

To achieve the above goals, we design a defense system which consists of two main components: the detection scheme and the locating scheme. The detection scheme provides in-network detection while the locating scheme provides exact locating. Both the detection scheme and locating scheme impose little computing overhead as well as communication overhead. The defense system as a whole is arbitrarily collusion resistant. Fig. 1 illustrates the overall structure of our defense system.

IV. KEY OBSERVATIONS AND APPROACH OVERVIEW

In this section, we describe our key observations and how we exploit these observations in our design of the detection and locating schemes.

A. In-Network Detection

Previous work that uses homomorphic MACs to detect corrupted packets, such as, the work by Agrawal and Boneh [10], Li *et al.* [11], and Zhang *et al.* [18], leverage the observation that if a packet does not belong to the source space, then it is a corrupted packet. The detection works by first establishing shared secret MAC keys between the source and the intermediate nodes. Then, using these secret keys, the source node can sign the fixed source space and the intermediate nodes can verify if their received packets belong to the source space.

Our detection scheme leverages a different observation: A packet sent by an intermediate node must belong to the space spanned by all packets that it received from its parents (Lemma 1). For example, consider a subset of nodes in a network in Fig. 2. At any moment in the multicast session, a packet sent by C must belong to the space spanned by the packets it received from its parents: A and B ; otherwise, C must be polluting the

network. Formally, at any moment t in the multicast session, if an intermediate node N sends out a vector \mathbf{y} then $\mathbf{y} \in \Pi_N(t)$; otherwise, \mathbf{y} is corrupted.

We use SpaceMac to enable the parents of N to sign the expanding space Π_N and the children of N to verify any packet that N sends to see if it belongs to Π_N . The cooperation of the parents of N and the children of N enables the children to detect any corrupted packet sent by N immediately. For example, in Fig. 2, at any time t , D is able to check if any packet it receives from C belongs to $\Pi_C(t)$; hence, D is able to detect any pollution attempt by C as soon as D receives a corrupted packet from C .

B. Exact Locating

Leveraging the cooperation between nodes in the network and a central controller, Jafarisiavoshani *et al.* [19] have shown that when there is a single pollution attacker, its location can be narrowed down to a set of at most two nodes. This is by analyzing the polluted edges identified based on the incoming subspaces reported by all the nodes to the central controller. An example is shown in Fig. 3. When there are multiple attackers in a general network topology, the number of suspected nodes increases to include the attackers and their parents and children.

Our key observation here is that the uncertainty about the location of the attackers originates from the fact that the attackers can lie about their received spaces. Therefore, by ensuring that all nodes in the network cannot lie about their received spaces, we can exactly locate the attackers. For instance, if the attacker cannot lie in the example given in Fig. 3, then the only possible scenario is scenario (2). Thus, one can determine that the attacker is at node B . We explicitly design and use SpaceMac to achieve this goal, *i.e.*, prevent nodes from lying. To prevent attacker B from lying, A cooperates with the controller by signing the space spanned by the packets it sends to B using SpaceMac. This is so that when B reports a fake space, it will not have the proper signature of the fake space to convince the controller.

V. THE CONSTRUCTION OF SPACEMAC

In this section, we describe the construction of SpaceMac. This construction is new and significantly more efficient than our old construction presented in [24]. This construction is an improvement of the homomorphic MAC construction, HomMac, proposed by Agrawal and Boneh [10]. The key difference between SpaceMac and HomMac is the security property that SpaceMac brings: SpaceMac *allows for signing spaces that expand over time, while HomMac only allows for signing fixed spaces*. This is directly reflected in the difference between the security games of SpaceMac and HomMac, and consequently the difference between the constructions. SpaceMac improves and generalizes of HomMac. Therefore, SpaceMac can also be used to sign fixed spaces as well, *e.g.*, in the place of HomMac in the detection scheme in [10]. However, it is not possible to use HomMac to support either our detection scheme or our locating scheme because any intermediate node does not know about the entire source space (to use Sign of HomMac) at the time of tag generation.

A. Definitions

A (q, n, m) homomorphic MAC scheme is defined by three probabilistic, polynomial-time algorithms: Mac, Combine, and Verify. The Mac algorithm generates a tag for a given vector; the Combine algorithm computes a tag for a linear combination of some given vectors; and the Verify algorithm verifies whether a tag is a valid tag of a given vector.

- **Mac**(k, id, \mathbf{y}):
 - Input: a secret key k , the identifier id of the source space Π^S , and a vector $\mathbf{y} \in \mathbb{F}_q^{n+m}$.
 - Output: tag t for \mathbf{y} .
- **Combine**(($\mathbf{y}_1, t_1, \alpha_1$), \dots , ($\mathbf{y}_p, t_p, \alpha_p$)):
 - Input: p vectors $\mathbf{y}_1, \dots, \mathbf{y}_p$, their tags t_1, \dots, t_p under key k , and their coefficients $\alpha_1, \dots, \alpha_p \in \mathbb{F}_q$.
 - Output: tag t for vector $\mathbf{y} \stackrel{\text{def}}{=} \sum_{i=1}^p \alpha_i \mathbf{y}_i$.
- **Verify**($k, \text{id}, \mathbf{y}, t$):
 - Input: a secret key k , the identifier id of the source space Π^S , a vector $\mathbf{y} \in \mathbb{F}_q^{n+m}$, and its tag t .
 - Output: 0 (reject) or 1 (accept)

Also, the scheme must satisfy the following correctness requirement:

$$\text{Verify}\left(k, \text{id}, \sum_{i=1}^p \alpha_i \mathbf{y}_i, \text{Combine}((\mathbf{y}_1, t_1, \alpha_1), \dots, (\mathbf{y}_p, t_p, \alpha_p))\right) = 1.$$

B. Attack Game

We consider the following attack game for a homomorphic MAC $\mathcal{T} = (\text{Mac}, \text{Combine}, \text{Verify})$, a challenger \mathcal{C} , and an adversary \mathcal{A} :

- **Setup**. \mathcal{C} generates a random key $k \xleftarrow{\mathcal{R}} \mathcal{K}$.
- **Queries**. \mathcal{A} adaptively queries \mathcal{C} , where each query is of the form (id, \mathbf{y}) . For each query, \mathcal{C} replies to \mathcal{A} with the corresponding tag $t \leftarrow \text{Mac}(k, \text{id}, \mathbf{y})$.
- **Output**. \mathcal{A} eventually outputs a tuple $(\text{id}_*, \mathbf{y}_*, t_*)$.

Up to the time \mathcal{A} outputs, it has queried \mathcal{C} multiple times. Let l denote the number of times \mathcal{A} queried \mathcal{C} using id_* and get tags for l vectors, $\mathbf{y}_{*1}, \dots, \mathbf{y}_{*l}$, of these queries. Let $\mathbf{y}_* = (y_{*1}^{(1)}, \dots, y_{*1}^{(n+m)})$. We consider that the adversary wins the security game if

- $(y_{*1}^{(n+1)}, \dots, y_{*1}^{(n+m)}) \neq \mathbf{0}$ (trivial forge otherwise),
- $\text{Verify}(k, \text{id}_*, \mathbf{y}_*, t_*) = 1$, and
- $\mathbf{y}_* \notin \text{span}(\mathbf{y}_{*1}, \dots, \mathbf{y}_{*l})$.

Let $\text{Adv}[\mathcal{A}, \mathcal{T}]$ denote the probability that \mathcal{A} wins the above attack game. We define a secure homomorphic MAC scheme as follows:

Definition 2. A (q, n, m) homomorphic MAC scheme \mathcal{T} is secure if for all probabilistic polynomial-time adversaries \mathcal{A} , $\text{Adv}[\mathcal{A}, \mathcal{T}]$ is negligible.

C. Construction

Let \mathcal{K} and \mathcal{D} denote the domains of the keys and the id 's of the spaces sent by the source, respectively. Let $[n]$ denote $\{1, \dots, n\}$. We use a pseudorandom generator (PRG) $G: \mathcal{K}_G \rightarrow \mathbb{F}_q^{n+m}$ and a PRF $F: \mathcal{K}_F \times \mathcal{D} \times [m] \rightarrow \mathbb{F}_q$. A

key k for this construction consists of a pair (k_1, k_2) , where $k_1 \in \mathcal{K}_G$ and $k_2 \in \mathcal{K}_F$.

- **Mac**(k, id, \mathbf{y}): A tag for a vector $\mathbf{y} = (y^{(1)}, \dots, y^{(n+m)})$ using key $k = (k_1, k_2)$ is generated as follows:
 - $\mathbf{r} \leftarrow G(k_1) \in \mathbb{F}_q^{n+m}$
 - $b \leftarrow \sum_{j=1}^m [y^{(n+j)} \cdot F(k_2, \text{id}, j)] \in \mathbb{F}_q$
 - $t \leftarrow (\mathbf{r} \cdot \mathbf{y}) + b \in \mathbb{F}_q$
- **Combine**(($\mathbf{y}_1, t_1, \alpha_1$), \dots , ($\mathbf{y}_p, t_p, \alpha_p$)): The tag t of \mathbf{y} is computed as follows:
 - $t \leftarrow \sum_{i=1}^p \alpha_i t_i \in \mathbb{F}_q$
- **Verify**($k, \text{id}, \mathbf{y}, t$): To verify if t is the valid tag of \mathbf{y} using key $k = (k_1, k_2)$, we proceed as follows:
 - $\mathbf{r} \leftarrow G(k_1) \in \mathbb{F}_q^{n+m}$
 - $b \leftarrow \sum_{j=1}^m [y^{(n+j)} \cdot F(k_2, \text{id}, j)] \in \mathbb{F}_q$
 - $a \leftarrow \mathbf{r} \cdot \mathbf{y} \in \mathbb{F}_q$
 - If $a + b = t$ output 1; otherwise, output 0

The correctness of the scheme is proved as follows. Suppose

$$\mathbf{y} = (y^{(1)}, \dots, y^{(n+m)}) = \sum_{i=1}^p \alpha_i \mathbf{y}_i.$$

Then,

$$\begin{aligned} a + b &= \mathbf{r} \cdot \sum_{i=1}^p \alpha_i \mathbf{y}_i + \sum_{j=1}^m \left[\left(\sum_{i=1}^p \alpha_i y_i^{(n+j)} \right) \cdot F(k_2, \text{id}, j) \right] \\ &= \sum_{i=1}^p \alpha_i \left[(\mathbf{r} \cdot \mathbf{y}_i) + \sum_{j=1}^m (y_i^{(n+j)} \cdot F(k_2, \text{id}, j)) \right] = \sum_{i=1}^p \alpha_i t_i. \end{aligned}$$

Compared to our old SpaceMac construction [24], this construction is significantly more efficient. In particular, compared to the old Mac and Verify algorithms, the new ones use one additional PRG call but significantly less number of PRF calls: m as opposed to $n + m$. Considering that a PRG computation is more efficient than a PRF computation and that in practice, n is typically an order of magnitude larger than m , e.g., $n = 2048, m = 128$ in a live video streaming system [25], this new construction is one order of magnitude more computationally efficient.

Compared to HomMac [10], our construction replaces the Sign algorithm, which generates tags for all basis vectors of a fixed space, with the Mac algorithm, which generates a tag for any vector in \mathbb{F}_q^{n+m} . The Combine algorithms show that tags generated by our Mac algorithms can be combined to produce a valid tag for an arbitrary linear combination. We note that both our constructions can be considered as a generalization of the scheme in [10] because the Mac algorithm can substitute the Sign algorithm, i.e., it can generate valid, combinable tags for all basis vectors.

D. Security

The security of SpaceMac is proven by assuming F is a secure PRF and G is a secure PRG. Let \mathcal{B}_1 and \mathcal{B}_2 denote a PRF adversary and a PRG adversary, respectively. Let $\text{PRF-Adv}[\mathcal{B}_1, F]$ and $\text{PRG-Adv}[\mathcal{B}_2, G]$ denote the advantages in winning the PRF and PRG security games, respectively.¹

¹The definition of PRF and PRG security games can be found in [26].

Theorem 3. For any fixed q, n, m , SpaceMac is a secure (q, n, m) homomorphic MAC assuming F is a secure PRF and G is a secure PRG. In particular, for every homomorphic MAC adversary \mathcal{A} , there is a PRF adversary \mathcal{B}_1 and a PRG adversary \mathcal{B}_2 , who have similar running time to \mathcal{A} , such that

$$\text{Adv}[\mathcal{A}, \text{SpaceMac}] \leq \text{PRF-Adv}[\mathcal{B}_1, F] + \text{PRG-Adv}[\mathcal{B}_2, G] + \frac{1}{q}.$$

Proof: The proof is by using a sequence of games denoted as Game 0, 1, and 2. Let W_0, W_1 , and W_2 denote the events that \mathcal{A} wins the homomorphic MAC security in Game 0, 1 and 2, respectively. Let Game 0 be identical to the Attack Game. Hence,

$$\Pr[W_0] = \text{Adv}[\mathcal{A}, \text{SpaceMac}]. \quad (1)$$

In Game 1, the PRG G is replaced by a truly random string, i.e., to respond to the Mac query, the challenger computes $\mathbf{r} \xleftarrow{R} \mathbb{F}_q^{n+m}$ instead of $\mathbf{r} \leftarrow G(k_1)$. Everything else remains the same. Then, there exists a PRG adversary \mathcal{B}_2 such that

$$|\Pr[W_0] - \Pr[W_1]| = \text{PRG-Adv}[\mathcal{B}_2, G]. \quad (2)$$

In Game 2, the PRF F is replaced by a truly random function, i.e., to respond to the Mac query, the challenger computes $b \leftarrow \sum_{j=1}^m [y^{(n+j)} \cdot s^{(j)}]$, where $s^{(j)} \xleftarrow{R} \mathbb{F}_q$ instead of $s^{(j)} \leftarrow F(k_2, \text{id}, j)$. Everything else remains the same. Then, there exists a PRF adversary \mathcal{B}_1 such that

$$|\Pr[W_1] - \Pr[W_2]| = \text{PRF-Adv}[\mathcal{B}_1, F]. \quad (3)$$

The complete challenger in Game 2 works as follows:

Initialization. $\mathbf{r} \xleftarrow{R} \mathbb{F}_q^{n+m}$

Queries. \mathcal{A} adaptively queries \mathcal{C} , where each query is of the form (id, \mathbf{y}) . \mathcal{C} replies to query i of \mathcal{A} as follows:

if id is never used in any of the previous queries:

$s_i^{(j)} \xleftarrow{R} \mathbb{F}_q$ for $j = 1, \dots, m$

else:

$s_i^{(j)}$'s := the ones used in the previous response

send $t := (\mathbf{r} \cdot \mathbf{y}) + \sum_{j=1}^m [y^{(n+j)} \cdot s^{(j)}]$

Output. \mathcal{A} eventually outputs a tuple $(\text{id}_*, \mathbf{y}_*, t_*)$. To determine if \mathcal{A} wins the game we compute

if $\text{id}_* = \text{id}_i$ (for some i) then // case (i)

set $s_*^{(j)} := s_i^{(j)}$ for $j = 1, \dots, m$

else // case (ii)

set $s_*^{(j)} \xleftarrow{R} \mathbb{F}_q$ for $j = 1, \dots, m$

Let l denote the number of times \mathcal{A} queried \mathcal{C} using id_* and get tags for l vectors, $\mathbf{y}_{*1}, \dots, \mathbf{y}_{*l}$, of these queries. We consider that the adversary wins the game, i.e., event W_2 happens, if

$$(y_*^{(n+1)}, \dots, y_*^{(n+m)}) \neq \mathbf{0}, \quad (4)$$

$$t_* = (\mathbf{r} \cdot \mathbf{y}_*) + \sum_{j=1}^m [y_*^{(n+j)} \cdot s_*^{(j)}], \text{ and} \quad (5)$$

$$\mathbf{y}_* \notin \text{span}(\mathbf{y}_{*1}, \dots, \mathbf{y}_{*l}). \quad (6)$$

In the following steps, we show that $\Pr[W_2] = \frac{1}{q}$. Let T be the event that \mathcal{A} outputs the tuple with a completely new id_* , i.e., \mathcal{A} never made queries using id_* before.

- When T happens, i.e., in case (ii), since $s_*^{(j)}$'s are indistinguishable from random values in \mathbb{F}_q , and $y_*^{(n+j)}$'s are not all zeros, the right hand side of equation (5) is a completely random value in \mathbb{F}_q , independent of the adversary's view. Thus,

$$\Pr[W_2 \wedge T] = \frac{1}{q} \Pr[T]. \quad (7)$$

- When T does not happen, i.e., in case (i), $s_*^{(j)}$ of equation (5) have been used to generate tags for vectors $\mathbf{y}_{*1}, \dots, \mathbf{y}_{*l}$. In this case, we will proceed by showing that when $\mathbf{y}_* \notin \text{span}(\mathbf{y}_{*1}, \dots, \mathbf{y}_{*l})$, the right hand side of equation (5) looks indistinguishable from a random value in \mathbb{F}_q . To this end, let $\Pi = \text{span}(\mathbf{y}_{*1}, \dots, \mathbf{y}_{*l})$, and d be the dimension of Π . Note that $d < n + m$ because otherwise $\Pi = \mathbb{F}_q^{n+m}$, which implies $\mathbf{y}_* \in \Pi$. Let $\{\mathbf{b}_1, \dots, \mathbf{b}_d\}$ be a basis of Π . Denote $\text{aug}(\mathbf{y})$ as the augmentation of vector \mathbf{y} , i.e., $\text{aug}(\mathbf{y}) = (y^{(n+1)}, \dots, y^{(n+m)})$.

Case (a): Consider the case when $\text{aug}(\mathbf{y}_*)$ can be expressed as a linear combination of $\text{aug}(\mathbf{b}_i), i \in [1, d]$. Let $\text{aug}(\mathbf{y}_*) = \sum_{i=1}^d \alpha_i \text{aug}(\mathbf{b}_i)$ for some α_i . If we let $\mathbf{y}' = \sum_{i=1}^d \alpha_i \mathbf{b}_i$, then the valid tag of \mathbf{y}' for the same space id_* as \mathbf{y}_* is

$$\begin{aligned} t' &= (\mathbf{r} \cdot \mathbf{y}') + \sum_{j=1}^m \left[\left(\sum_{i=1}^d \alpha_i \mathbf{b}_i \right)^{(n+j)} \cdot s_*^{(j)} \right] \\ &= (\mathbf{r} \cdot \mathbf{y}') + \sum_{j=1}^m [y_*^{(n+j)} \cdot r_*^{(j)}] \end{aligned} \quad (8)$$

By subtracting equation (8) from (5), we know that by producing a valid forgery, the adversary found a \mathbf{y}_* and t_* that satisfy the following equation:

$$t_* - t' = \mathbf{r} \cdot (\mathbf{y}_* - \mathbf{y}') \quad (9)$$

However, since $\mathbf{y}_* \neq \mathbf{y}'$ (\mathbf{y}' is in Π but \mathbf{y}_* is not), and \mathbf{r} is indistinguishable from a random vector in \mathbb{F}_q^{n+m} , the probability that he can satisfy (9) is exactly $\frac{1}{q}$.

Case (b): Consider the case when $\text{aug}(\mathbf{y}_*)$ cannot be expressed as a linear combination of $\text{aug}(\mathbf{b}_i)$'s. In this case, we proceed by showing that given a fixed \mathbf{y}_* from the perspective of the adversary, the valid tag t_* of \mathbf{y}_* is indistinguishable from a random value in \mathbb{F}_q :

Let $s_*^{(i)}, i \in [1, m]$, be the unknowns, and $\mathbf{s} = (s_*^{(1)}, \dots, s_*^{(m)})$. By the previous l queries, the adversary learns the following system of l equations and m unknowns:

$$(I) \begin{cases} \text{aug}(\mathbf{y}_{*1}) \cdot \mathbf{s} = t_{\mathbf{y}_{*1}} - \mathbf{r} \cdot \mathbf{y}_{*1} \\ \dots \\ \text{aug}(\mathbf{y}_{*l}) \cdot \mathbf{s} = t_{\mathbf{y}_{*l}} - \mathbf{r} \cdot \mathbf{y}_{*l} \end{cases}$$

Since $\{\mathbf{b}_1, \dots, \mathbf{b}_d\}$ is a basis of $\Pi = \text{span}(\mathbf{y}_{*1}, \dots, \mathbf{y}_{*l})$, the above system is equivalent

to the following system of d equations:

$$(II) \begin{cases} \text{aug}(\mathbf{b}_1) \cdot \mathbf{s} = u_1 \\ \dots \\ \text{aug}(\mathbf{b}_d) \cdot \mathbf{s} = u_d \end{cases}$$

where each u_j is a linear combination of right-hand-side values of the equations of system (I).

Let $\Pi' = \text{span}(\text{aug}(\mathbf{b}_1), \dots, \text{aug}(\mathbf{b}_d))$ and d' be the dimension of Π' . Note that $d' \leq d$. Let $\{\mathbf{c}_1, \dots, \mathbf{c}_{d'}\}$ be a basis of Π' . Note that since $\text{aug}(\mathbf{y}_*) \notin \Pi'$, $\text{aug}(\mathbf{y}_*)$ cannot be expressed as a linear combination of \mathbf{c}_i 's. The system of equations (II) is equivalent to the following system of d' equations:

$$(III) \begin{cases} \mathbf{c}_1 \cdot \mathbf{s} = e_1 \\ \dots \\ \mathbf{c}_{d'} \cdot \mathbf{s} = e_{d'} \end{cases}$$

where each e_j is a linear combination of right-hand-side values of the equations of system (II). A valid tag t_* of \mathbf{y}_* satisfies the following equation:

$$\text{aug}(\mathbf{y}_*) \cdot \mathbf{s} = t_* - \mathbf{r} \cdot \mathbf{y}_*. \quad (10)$$

Without loss of generality, assume that the adversary knows \mathbf{r} . Note that $d' < m$ otherwise $\text{aug}(\mathbf{y}_*) \in \text{span}(\mathbf{b}_i)$. Since $d' \leq d$, it follows that $d' < m$. The system of m unknowns and $d' + 1$ linear equations, d' from (III) and 1 from (10), is consistent regardless of the value of t_* because the coefficient matrix, whose rows are linearly independent vectors: $\mathbf{c}_1, \dots, \mathbf{c}_{d'}$, and $\text{aug}(\mathbf{y}_*)$, has rank $d' + 1 \leq m$. Furthermore, for any value t_* , the solution space always has size $q^{m-d'-1}$. Thus, for a fixed \mathbf{y}_* , its valid tag, t_* , could be any value in \mathbb{F}_q equally likely, given that $s_*^{(i)}$'s are chosen uniformly at random from \mathbb{F}_q . Hence, the probability of forging a valid tag t_* is $\frac{1}{q}$.

By the result of case (a) and case (b),

$$\Pr[W_2 \wedge \neg T] = \frac{1}{q} \cdot \Pr[\neg T]. \quad (11)$$

- From equations (7) and (11), we have

$$\Pr[W_2] = \Pr[W_2 \wedge T] + \Pr[W_2 \wedge \neg T] = \frac{1}{q}. \quad (12)$$

Equations (1), (2), (3), and (12) together prove the theorem. ■

Theorem 3 states that an adversary \mathcal{A} can break the scheme with probability $\frac{1}{q}$. For a small field size, e.g., $q = 2^8$, the security of the MAC scheme may be unsatisfactory. To improve the security, one could either increase the field size or use multiple tags as suggested in [10] and [11]. The security of our scheme using l tags is $(\frac{1}{q})^l$. As observed in [11], it is preferable to use multiple tags instead of increasing the field size. This is because in order to achieve the same security $(\frac{1}{q})^l$, using the field size q^l instead of using l tags increases the computational complexity of field multiplication by $\log l$ times.

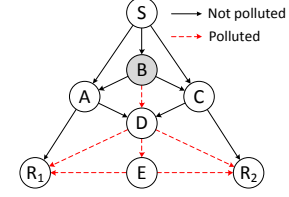


Fig. 4. A network consisting of 8 nodes (resembling a network given in [19]). B is the attacker. After every node (except for S) reports to the controller its true incoming spaces, B is identified as the attacker since it has no incoming polluted edge but has one outgoing polluted edge.

VI. DETECTION SCHEME

In this section, we describe our detection scheme in detail. Our scheme exploits the observation outlined in Section IV-A to provide in-network detection. In particular, parents and children of an intermediate node N cooperate through SpaceMac to detect any corrupted packet sent by N in a timely manner. For ease of presentation, we describe the detection scheme within the scope of a single generation, i.e., considering a single source space id.

1) Assumptions. We assume that there is a controller who knows the complete topology of the graph. The controller could be the source itself. This assumption is also made in recent work by Li *et al.* [11]. We further assume that each node N shares with the controller a pair of secret keys (k_N^1, k_N^2) . These keys can be established with the help of a Public Key Infrastructure (PKI). We note that recent proposed schemes which use homomorphic MACs also made assumption about the existence of a PKI [11], [18] or the existence of shared secret keys [10]. In general, the problem of establishing shared secret keys is a challenging problem of its own and is orthogonal to this work.

2) Bootstrapping. First, for every intermediate node N , the controller determines the key $k_{\bar{N}}$, which will be secret to N itself and is used by the parents and children of N when using SpaceMac. Each node can serve as either a parent or a child; therefore, each node, depending on its position in the network, requires to know a different set of keys to participate in the detection scheme. For example, consider the network in Fig. 4, node D needs to know $k_{\bar{A}}, k_{\bar{B}}$, and $k_{\bar{C}}$ to detect corrupted packets sent by A, B and C , respectively. It also needs to know $k_{\bar{E}}$ to help R_1 and R_2 to detect corrupted packets sent by E .

Second, the source and all the receivers need to share an end-to-end key, k^* . This key is used to ensure detection in the presence of colluding adversaries, in which case a node N colludes with its parent to obtain $k_{\bar{N}}$ and thus can bypass the verification of its children. We defer the discussion about multiple adversaries to Section VIII-B, where we analyze different colluding scenarios in depth.

The controller then sends to each node N a bootstrapping packet consisting of the set of keys that are necessary for it to participate in the detection scheme. In particular, the controller constructs the bootstrapping packet b_N : $b_N = \{k_{\bar{X}}, X \in \{\mathcal{P}_N \cup \mathcal{C}_N\}, k^*\}$. Note that b_N contains k^* if and only if N is either the source or a receiver. The controller then sends b_N to N through a secure and authenticated channel achieved

with k_N^1 (for encryption) and k_N^2 (for authentication) using a standard encrypt-then-authenticate algorithm². For example, node D in Fig. 4 receives $\{k_{\bar{A}}, k_{\bar{B}}, k_{\bar{C}}, k_{\bar{E}}\}$ while node R_1 receives $\{k_{\bar{A}}, k_{\bar{D}}, k_{\bar{E}}, k^*\}$.

We note that the MAC keys k^* and $k_{\bar{X}}$ can be used for multiple source space/generations. This is because, by construction, SpaceMac takes into account generation identifiers when computing tags. As a result, the overhead of a key establishment is per multiple generations as opposed to per single generation. Hence, this overhead is asymptotically negligible in the number of generations.

Finally, if each node knows its own 2-hop neighborhood information, then the MAC keys can also be bootstrapped without the help of the controller. In particular, by using a secure key distribution scheme for ad-hoc networks, such as [27], the source can establish the shared secret MAC key k^* with the receivers, and for a node N , a parent P can establish a shared secret MAC key $k_{\bar{N}}$ with the parents and children of N .

3) Sending and Coding. Before sending out the each source packet, \mathbf{v}_i , the source S calculates an *end-to-end tag*, $t_{\mathbf{v}_i}^{k^*}$, using the Mac algorithm of SpaceMac with key k^* : $t_{\mathbf{v}_i}^{k^*} = \text{Mac}(\mathbf{v}_i, k^*)$. S then attaches this tag to every source packet and sends $\mathbf{w}_i \triangleq \{t_{\mathbf{v}_i}^{k^*} || \mathbf{v}_i\}$ instead of \mathbf{v}_i , where ‘||’ denotes concatenation. The packets traversing the network are linear combinations of \mathbf{w}_i ’s instead of \mathbf{v}_i ’s. For ease of presentation with regards to the input length of the SpaceMac algorithms, assume that the size of \mathbf{w}_i is still n .

Consider a parent P who wants to send a packet \mathbf{y} to its child N . P needs to calculate a *helper tag* which helps the children of N to detect corrupted packets sent by N . In particular, before sending \mathbf{y} to N , P needs to calculate a MAC tag, $t_{\mathbf{y}}^{k_{\bar{N}}}$, using Mac under key $k_{\bar{N}}$: $t_{\mathbf{y}}^{k_{\bar{N}}} = \text{Mac}(\mathbf{y}, k_{\bar{N}})$. Besides the helper tag, P must also pass along a *verification tag* of \mathbf{y} , which is used by N to verify the integrity of \mathbf{y} . Assume P received $\{\mathbf{y}_1, \dots, \mathbf{y}_l\}$ and their helper tags $\{t_{\mathbf{y}_1}^{k_{\bar{P}}}, \dots, t_{\mathbf{y}_l}^{k_{\bar{P}}}\}$ from its parents, and P computes \mathbf{y} as $\mathbf{y} = \sum_{i=1}^l \alpha_i \mathbf{y}_i$. Then, the verification tag, $t_{\mathbf{y}}^{k_{\bar{P}}}$, of \mathbf{y} can be computed using Combine: $t_{\mathbf{y}}^{k_{\bar{P}}} = \text{Combine}((\mathbf{y}_1, t_{\mathbf{y}_1}^{k_{\bar{P}}}, \alpha_1), \dots, (\mathbf{y}_l, t_{\mathbf{y}_l}^{k_{\bar{P}}}, \alpha_l))$. The final packet that P sends to N includes \mathbf{y} and its helper and verification tags: $\{t_{\mathbf{y}}^{k_{\bar{N}}} || t_{\mathbf{y}}^{k_{\bar{P}}} || \mathbf{y}\}$.

We note that if a node is benign, besides explicitly calculating the helper tag, it would code and send packets in a way identical to what it does normally. The verification tag will be computed correctly because the Combine algorithm linearly combines the tags in the same way the packets are combined, i.e., with the same set of coefficients.

4) Receiving and Verification. When a node N receives from its parent P a packet $\{t_{\mathbf{y}}^{k_{\bar{N}}} || t_{\mathbf{y}}^{k_{\bar{P}}} || \mathbf{y}\}$, it uses $k_{\bar{P}}$ and the Verify algorithm to verify the integrity of the packet. The packet is deemed non-corrupted if $\text{Verify}(k_{\bar{P}}, \mathbf{y}, t_{\mathbf{y}}^{k_{\bar{P}}}) = 1$. The security guarantee comes from the security of SpaceMac: since P does not know $k_{\bar{P}}$, the probability that P can forge a valid tag, $t_{\mathbf{y}}^{k_{\bar{P}}}$, when \mathbf{y} is outside of its received space, Π_P , is $\frac{1}{q}$. As a result, as soon as N receives a corrupted packet

from P , with high probability, N is able to detect the attack immediately.

In the case N is a receiver, it further verifies the end-to-end tag using key k^* . Parse \mathbf{y} as $\{t_{\mathbf{w}}^{k^*} || \mathbf{w}\}$. N accepts \mathbf{w} if $\text{Verify}(k^*, \mathbf{w}, t_{\mathbf{w}}^{k^*}) = 1$. The security guarantee, again, comes from the security of SpaceMac: since none of the malicious intermediate node knows k^* , if \mathbf{w} is outside of the source space, the adversary can only forge a valid tag of \mathbf{w} , $t_{\mathbf{w}}^{k^*}$, with a negligible probability of $\frac{1}{q}$. This second level of verification is to provide a detection mechanism in the presence of colluding adversaries.

VII. LOCATING SCHEME

Locating the attackers to eventually eliminate them is a logical step after a pollution attack is detected. In this section, we describe in detail how we exploit the observation made in Section IV-B to exactly locate the pollution attackers. The main idea is to force nodes to truthfully report their received spaces to correctly identify polluted edges, thereby enabling the exact identification of the location of the attackers.

A. Overview

1) Reporting. The following lemma, originally presented in [28] and [19], implies that for each received subspace, Π_N^P , from a parent P , node N may report a randomly chosen packet, \mathbf{y}_r , of the space instead of the space itself; and by checking if $\mathbf{y}_r \in \Pi^S$, the controller can determine if $\Pi_N^P \subseteq \Pi^S$ to identify the polluted edges.

Lemma 4 (Jafarisiavoshani *et al.* [19], [28]). *Let Π_1 and Π_2 be two subspaces of \mathbb{F}_q^{n+m} and assume that \mathbf{y}_r is a randomly selected packet from Π_1 . Let d_{12} and d_1 are the dimensions of $\Pi_1 \cap \Pi_2$ and Π_1 , respectively. With probability $1 - q^{d_{12}-d_1}$, $\mathbf{y}_r \in \Pi_2$ if and only if $\Pi_1 \subseteq \Pi_2$.*

2) Using SpaceMac. We use SpaceMac to prevent nodes from lying about their received spaces as follows. To enforce a node N to report a true received space, Π_N^P , that it received from its parent, P , the parent P and the controller cooperate so that the controller only accepts reported packets belonging to but not outside of Π_N^P . In particular, whenever P sends a vector \mathbf{y}_i to N , it generates a tag, $t_{\mathbf{y}_i}$, of \mathbf{y}_i using the Mac algorithm with a secret key shared by P and the controller. Then, when N reports \mathbf{y}_r , if \mathbf{y}_r is a linear combination of vectors that it received from P , \mathbf{y}_i ’s, then N can generate a valid tag for \mathbf{y}_r by using the Combine algorithm on the tags of \mathbf{y}_i ’s that it received; if \mathbf{y}_r is not a linear combination of \mathbf{y}_i ’s then N can forge a valid tag for \mathbf{y}_r with only a negligible probability of $\frac{1}{q}$.

3) Non-Repudiation Transmission Protocol. As presented, SpaceMac forces nodes to report only true received subspaces since it is computationally difficult to forge valid tags otherwise. However, it does not prevent a malicious node from sending invalid tags to its children to prevent the children from reporting polluted spaces.

For example, an attacker P can send a polluted packet $\mathbf{y}_e \notin \Pi^S$ and a bogus tag t_e to its child N . When N reports the space Π_N^P , if the randomly chosen vector \mathbf{y}_r was formed by

²We refer the reader to Chapter 4.9 in [26] for more details on encrypt-then-authenticate algorithms.

a linear combination involving y_e , then the aggregated tag t_r of y_r that N generates using the Combine algorithm will be invalid due to the bogus tag t_e . As a result, the controller will reject y_r . Consequently, the attacker P successfully prevents its benign child N from reporting the polluted space Π_N^P .

To address this, we utilize an efficient non-repudiation transmission protocol proposed by Wang *et al.* [20]. For a parent P and a child N , the controller generates a set of secret keys, denoted by \mathcal{X} , based on the private key of the parent and the ID N of the child. After that, the controller randomly selects a set of keys \mathcal{Y} from \mathcal{X} based on the private key of the child and the ID P of the parent; then, it sends \mathcal{Y} to the child. We denote $\mathcal{X} \setminus \mathcal{Y}$ as $\overline{\mathcal{Y}}$; also, let $\lambda \triangleq |\mathcal{X}|$ and $\delta \triangleq |\mathcal{Y}|$.

When sending a packet, P generates λ tags (instead of one) using the Mac algorithm and all keys in \mathcal{X} . When receiving a packet, N uses its set of keys \mathcal{Y} and the Verify algorithm to verify δ out of λ tags. Finally, when receiving a randomly chosen packet y_r chosen from Π_N^P and its λ tags from the N , the controller uses all keys in $\overline{\mathcal{Y}}$ and the Verify algorithm to verify all $\lambda - \delta$ tags. The controller, in this case, keeps track of a counter θ , $\theta \leq \lambda - \delta$. If at least θ tags pass the verification then the controller accepts the report.

The following two lemmas provide the security of the non-repudiation transmission protocol when applying to our context. Lemma 5 is identical to Theorem 1 in [20]. Lemma 6 is an adapted version of Theorem 2 in [20] – the difference is that in our case, a node does not report a packet that it receives, but it reports a linear combination of packets that it receives instead.

Lemma 5 (Non-repudiation of the receiver–Wang *et al.* [20]). *The probability that a malicious child node can successfully report to the controller that its parent sends it a packet y , which is never sent by the parent, is at most*

$$\sum_{i=\theta}^{\lambda-\delta} \binom{\lambda-\delta}{i} \frac{1}{q^i} \left(1 - \frac{1}{q}\right)^{\lambda-\delta-i}.$$

Lemma 6 (Non-repudiation of the sender–Wang *et al.* [20]). *The probability that a malicious parent can make the controller reject the parent's space reported by its child by sending the child some packets with invalid tags is at most*

$$\max_{0 \leq x \leq \delta + \theta - 1} p(x), \text{ where } p(x) \leq \sum_{i=\max(x-\theta+1, 0)}^{\min(\delta, x)} \frac{\binom{\delta}{i} \binom{\lambda-\delta}{x-i}}{\binom{\lambda}{x} q^{\delta-i}}.$$

The proofs of these two lemmas are provided in the Appendix. Finally, we note that both of the above probabilities can be made very small by choosing appropriate values for q , λ , δ , and θ . Examples of values for these parameters and the corresponding probabilities are provided in Table III. The choice of parameters can then be made based on the desired tradeoff between the overhead and the probability that the attacker succeeds.

4) Locating the Attackers. After the controller collects the true subspaces from every node, we proceed similar to the approach by Jafarisiavoshani *et al.* [19] to locate the attackers. Here, we discuss the case when there is a single attacker. We

defer the case when there are multiple attackers to Section VIII-B.

In [19], the authors have shown that in a general network which has a single adversary, the location of the adversary can be narrowed down to a set of at most two nodes in both cases where the adversary inject corrupted packets to either one downstream edge or multiple downstream edges. This is done by partitioning the edges into two set: the set of polluted edges, \mathcal{E}_p , and non-polluted edges, \mathcal{E}_s , then analyzing the nodes with respect to the identified \mathcal{E}_p and \mathcal{E}_s . They also note that the partitioning of \mathcal{E}_p and \mathcal{E}_s is not unique since the adversary might lie, which results in the uncertainty about the location of the attacker.

Fortunately, when the partition reflects the real state of pollution of the edges in the network, *i.e.*, when the adversary is forced to report its true incoming spaces, the adversary is always the node that has no incoming edge belonging to \mathcal{E}_p but has at least one outgoing edge belonging to \mathcal{E}_p . Fig. 4 shows the case where B is an attacker whom get identified because it has no incoming polluted edge but one outgoing polluted edge.

Using our scheme, the probability that the attacker lies about its incoming spaces is very small (Lemma 5). Furthermore, the probability that the attacker can prevent its children from reporting the subspaces polluted by itself is very small, too (Lemma 6). As a result, with high probability (depending on q , λ , δ , and θ), our scheme can produce an unambiguous partitioning of \mathcal{E}_p and \mathcal{E}_s , which helps to precisely locate the attacker.

B. Full Description

To distinguish cryptographic keys used in the detection scheme and keys used in the locating scheme, we decorate any key used in the locating scheme with an overhead bar, *e.g.*, \bar{k} .

1) Assumptions. Similar to the assumptions we made in the detection scheme, we assume that there is a controller (could be the source itself) who knows the complete topology and the source space. This assumption is also made in recently proposed locating schemes [19], [20]. We assume that each node N shares a triplet of secret keys $(\bar{k}_N^1, \bar{k}_N^2, \bar{k}_N^3)$ with the controller (with the help of a PKI). We further assume that each node knows the identifiers of its adjacent nodes (can be bootstrapped by the controller). In addition, we assume that there is a reliable low-bandwidth end-to-end communication path between the controller and each node (the channel for the reports and the announcement by the controller). Other locating schemes, such as, [19] and [20], implicitly made this assumption.

2) Bootstrapping. Let \mathcal{N} be the set of IDs of adjacent downstream nodes of P . For $N \in \mathcal{N}$, the controller generates a set \mathcal{X}_{PN} of λ keys using a PRF $F_1: \mathcal{K} \times \mathcal{I} \times [\lambda] \rightarrow \mathcal{K}$, where \mathcal{K} is the domain of key \bar{k}_P^1 , and \mathcal{I} is the domain of the identifiers of the nodes: $\mathcal{X}_{PN} \leftarrow \{F_1(\bar{k}_P^1, N, i), \text{ for } i = 1, \dots, \lambda\}$. Note that P can compute \mathcal{X}_{PN} itself as it knows \bar{k}_P^1 and its neighbors' identifiers.

For $N \in \mathcal{N}$, consider an array L whose elements are distinct subsets of size δ of \mathcal{X}_{PN} . Note that L has length $\binom{\lambda}{\delta}$. The

controller uses another PRF $F_2: \mathcal{K} \times \mathcal{I} \rightarrow [\binom{\lambda}{\delta}]$ to select from L a subset of size δ : $\mathcal{Y}_{PN} = L[i]$, where $i \leftarrow F_2(\bar{k}_N^1, P)$. The controller then sends \mathcal{Y}_{PN} to node N through a secure and authenticated channel achieved with \bar{k}_N^2 (for encryption) and \bar{k}_N^3 (for authentication) using an encrypt-then-authenticate algorithm. Note that similar to k^* and $k_{\bar{X}}$, the sets of keys \mathcal{X}_{PN} and \mathcal{Y}_{PN} can be used across multiple generations. Denote $\mathcal{X}_{PN} \setminus \mathcal{Y}_{PN}$ as $\bar{\mathcal{Y}}_{PN}$.

3) Sending and Receiving. Let id be the identifier of the current source space Π^S . When a node P sends a packet \mathbf{y} to its downstream node N , beside the id , it has to send along λ tags, which are computed using the Mac algorithm and keys in \mathcal{X}_{PN} . Let $\mathcal{G}_{PN}(\mathbf{y})$ denote this set of tags. Node P sends $(id, \mathbf{y}, \mathcal{G}_{PN}(\mathbf{y}))$. When node N receives this packet from node P , it uses \mathcal{Y}_{PN} and the Verify algorithm to check the validity of δ out of λ tags of $\mathcal{G}_{PN}(\mathbf{y})$. It drops \mathbf{y} as long as there is an invalid tag. Otherwise, it stores the received tuple in its buffer.

4) Pollution Detection and Alert. A detection of the pollution is needed to start the locating process. Here, we use our detection scheme to provide the detection. Nevertheless, we stress that our locating scheme does not depend on any particular detection scheme. Using our detection scheme, a node N , upon detecting a pollution, sends an alert ($id \parallel N$) to the controller through an authenticated channel achieved using a traditional MAC scheme, *e.g.*, HMAC, and shared key \bar{k}_N^3 . When the controller receives an alert, it determines if id is reported before, if so, it ignores the alert. Otherwise, it sends a request (id) to each node N through an authenticated channel achieved with HMAC and \bar{k}_N^3 . This request demands each node to report its incoming subspaces.

5) Reporting Subspaces. Upon receiving the request (id) from the controller, each node N checks if it receives a similar request for the same id before, if it does, it ignores the request. Otherwise, it prepares the report as follows: For each parent node P , let $(\mathbf{y}_1, t_{1,1}, \dots, t_{1,\lambda}), \dots, (\mathbf{y}_l, t_{l,1}, \dots, t_{l,\lambda})$ be packets of source space id and their tags that node N received from node P . Node N sends to the controller through an authenticated channel achieved with HMAC and \bar{k}_N^3 the report $(N \parallel P \parallel \mathbf{y}_r \parallel t_1 \parallel \dots \parallel t_\lambda)$, where $\alpha_i \xleftarrow{R} \mathbb{F}_q$ ($i \in [l]$); $\mathbf{y}_r = \sum_{i=1}^l \alpha_i \mathbf{y}_i$; $t_j = \sum_{i=1}^l \alpha_i t_{i,j}$ ($j \in [\lambda]$).

6) Locating the Attackers. After sending out the requests, the controller waits for the reports. After Δt seconds, it starts identifying the pollution attackers. It classifies any node that does not report all of its incoming spaces as a malicious node. It only accepts reports with at least θ valid SpaceMac tags, where the validation uses keys in $\bar{\mathcal{Y}}_{PN}$'s. It then identifies the polluted edges in the network based on the reported spaces and the source space. We note that checking if a reported space is polluted can be done quickly and efficiently in $O(mn)$ in terms of multiplication operations by leveraging the global coding coefficients of the reported packet and the source packets. Finally, any node that does not have a polluted incoming edge but has a polluted outgoing edge is classified as malicious.

7) Releasing the Result. After identifying the set of attackers \mathcal{A} , the controller sends (\mathcal{A}) to each benign node N through an authenticated channel achieved with HMAC and key \bar{k}_N^3 . Upon receiving (\mathcal{A}) , each node N adds the nodes in \mathcal{A}

into its blacklist. Every node in the network will neither send nor receive traffic from nodes in its blacklist in subsequent communication. The controller itself removes nodes in \mathcal{A} as well as incoming and outgoing edges of these nodes from the network topology. Note that the MAC keys used in the detection scheme do not need to be refreshed when a node is removed from the network. This is because the parent-child relationship between any pair of the remaining adjacent nodes is the same as before the removal.

VIII. SECURITY ANALYSIS

A. Single Adversary

We described how our our detection and locating scheme work when there is a single adversary when we describe our schemes. We refer the reader to Section VI and Section VII for the details.

B. Multiple Adversaries

1) Detection Scheme:

a. Independent Adversaries: We consider adversaries as independent when every adversary checks the integrity of the packets it receives from its parents and drop corrupted packets. This scenario is similar to the single adversary scenario. In-network detection works because if a node N wants to pollute the network, it still has to forge a valid SpaceMac verification tag for a packet that lies outside of its received space Π_N , which is computationally difficult.

b. Colluding Adversaries: We first consider the *passive* colluding scenario, where there is an adversary who does not drop corrupted packets that it receives from its parents. In this scenario, in-network detection no longer works. To see this, consider the case where N receives from one of its parents, P , a corrupted packet $\{t_{\mathbf{y}}^{k_N} \parallel t_{\mathbf{y}}^{k_P} \parallel \mathbf{y}\}$ with correct helper tag, $t_{\mathbf{y}}^{k_N}$, and invalid verification tag, $t_{\mathbf{y}}^{k_P}$, and N does not drop this packet. To propagate the pollution to its child C , it simply computes an appropriate helper tag, $t_{\mathbf{y}}^{k_C}$, for this packet using k_C , then forward the packet $\{t_{\mathbf{y}}^{k_C} \parallel t_{\mathbf{y}}^{k_N} \parallel \mathbf{y}\}$ to C . Clearly, $t_{\mathbf{y}}^{k_N}$ is a valid tag for \mathbf{y} ; hence, \mathbf{y} passes C verification, and $t_{\mathbf{y}}^{k_C}$ is a valid helper tag for \mathbf{y} which will pass any verification by a child of C .

We now consider the *active* colluding scenario, where a node N can collude with one of its parents, P , to learn about the private key, k_N , that is used for verification by its children. In this scenario, in-network detection also fails. This is because knowing the secret key, k_N , N can generate a valid verification tag for any packet outside of its received space Π_N ; thus, any child C of N would not be able to detect corrupted packets sent by N .

Note that in both cases where the in-network detection fails, the adversaries must be *adjacent* to each other. In both of these cases, the end-to-end detection made by the receivers comes to the rescue. This end-to-end detection is reliable because the receivers are trusted and the private key k^* shared by the source and the receivers is not known to any adversary. In order to pass the verification done by the receivers, an adversary has to forge a valid SpaceMac tag

which is computationally difficult. We discuss how we could relax the assumption of trusted receivers in Section VIII-E.

2) Locating Scheme:

In the presence of multiple adversaries, an attacker may be “in the shadow” of some other attackers, which means that it may pollute only already polluted data and thus does not produce any detectable effect. More precisely, we define shadowed and exposed attackers below.

Definition 3 (Adapted from [19] and [20]). *An attacker is shadowed if it has at least one polluted incoming edge and is exposed otherwise.*

a. Independent Adversaries: In this case, we note that with high probability, our approach is already able to identify all exposed attackers. We utilize the following observation to identify all shadowed and exposed attackers.

Lemma 7. *For any directed acyclic graph with pollution attack in presence, there is at least one exposed attacker.*

Proof: Consider a topological ordering of the graph, the first malicious node in the ordering is an exposed attacker. ■

Exploiting this, we can use multiple generations, *i.e.*, transmissions of (different) source spaces, to identify all attackers.

Lemma 8. *In a network with η independent attackers. With high probability (depending on q , λ , δ , and θ), all attackers can be identified after κ generations which experience pollution attack, where $\kappa \leq \eta$.*

Proof: Since there is at least one exposed attacker per generation by Lemma 7, our scheme can identify at least one attacker per generation. Because the identified attackers are immediately excluded from future communication and the other attackers are persistent, subsequent identified attackers are different from the already identified ones. Therefore, it takes at most η generations to identify all η attackers. ■

Note that we consider the cases where there exists an attacker who is disconnected from the receivers after the removal of all other attackers as degenerate cases. This is because the disconnected attacker is no longer able to pollute the network. In this case, the location of all attackers cannot be determined.

b. Colluding Adversaries: We note that each pair of parent P and child N uses distinct key sets \mathcal{X}_{PN} and \mathcal{Y}_{PN} ; thus, the collusion of malicious nodes does not provide knowledge about the key sets of benign nodes. However, when the distance between any two attackers equals to one, where distance refers to the length of the shortest path connecting two nodes, these attackers can collude to report a false space.

Assume that in a network, there are colluding attackers P and N connected by a directed edge $e(P, N)$ and there is no other pair of attackers in the network having distance one. We ask the question: “What can P and N achieve by manipulating edge $e(P, N)$?” Consider a topological ordering \mathcal{O} of the nodes. If N makes $e(P, N) \in \mathcal{E}_p$, the set of polluted edges identified by the controller, then P will be exposed and identified after all malicious nodes that come before P in \mathcal{O} are identified. After P is located, N and the rest of the attackers will be eventually located. Otherwise, if N

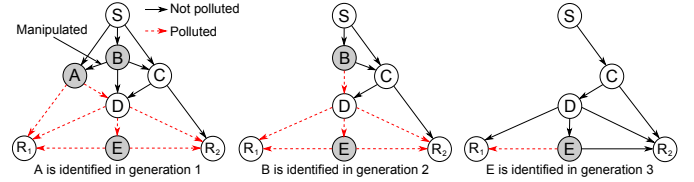


Fig. 5. An example where there are three attackers A , B , and E . Attackers A and B collude to make edge $e(B, A)$, which is polluted, non-polluted. Nevertheless, all are identified after 3 generations.

makes $e(P, N) \in \mathcal{E}_s$, the set of non-polluted edges identified by the controller, then N will be exposed and located after all malicious nodes that come before N in \mathcal{O} are located. Analogous to the other case, after N is located, P (if not already located) and the rest of the attackers will be eventually located. Consequently, by manipulating the status of edge $e(P, N)$, the attackers can, at best, change the order in which P and N are located. The above analysis can be extended to the general case where there are multiple pairs having distances one by considering the pair (P, N) , where N has a polluted outgoing edge, that appears first in \mathcal{O} first. Fig. 5 shows an example. As a result, we can generalize lemma 8:

Lemma 9. *In a network with η attackers. With high probability (depending on q , λ , δ , and θ), all attackers can be identified after κ generations which experience pollution attack, where $\kappa \leq \eta$.*

C. Tag-Pollution Resistance

As pointed out by Li *et al.* [11], a scheme that uses multiple MAC tags, such as [10], [17], may suffer from *tag pollution* attacks. In these schemes, a packet carries multiple tags and each node only has keys to verify a subset of them; therefore, an adversary may tamper with some of the tags which only get verified far down the information flow. The consequence is that a packet with some corrupted tags may pass the verification of a few level of nodes. When mixed with other packets, one corrupted tag may snowball into a large number of corrupted tags. The packets carrying these corrupted tags eventually fail the authentication down the stream, thus wasting resources of the network. This effectively emulates a pollution attack.

Both our detection scheme and locating scheme use multiple MAC tags; fortunately, our schemes are resistant to tag pollution attack. More specifically, in our detection scheme, each packet carries three tags: one end-to-end tag, one helper tag, and one verification tag. An adversary cannot tamper with the end-to-end tag because the verification tag of a packet is computed over the concatenation of both the content of the packet and its end-to-end tag. In other words, if the adversary tampers with the end-to-end tag of a packet, the packet will not pass the verification test made by the immediate downstream nodes. This idea of using nested tags was originally introduced in [11]. Apparently, the adversary cannot tamper with the verification tag of a packet since this verification tag is checked immediately by one of its children. Finally, the adversary may tamper with the helper tag of a packet it sends to its child, *e.g.*, attaches an erroneous helper tag. In this case, since the child uses the helper tag to compute verification tags for its

outgoing packets, any packet involving the packet with an erroneous helper tag, sent by the child, will have a corrupted verification tag. The next hop that receives packets from this child will drop any packet with a corrupted verification tag. Note that no other tags, besides the end-to-end tags, travel more than two hops in our detection scheme. This eliminates the scenarios where tags are only verified far down the stream as in a tag pollution attack. Our locating scheme uses λ tags; however, these tags are never forwarded to any next hop other than the controller (only during the locating process). Furthermore, Lemma 6 ensures that an adversary can only trick the controller by sending erroneous tags with negligible probability. As a result, our locating scheme is also not susceptible to tag pollution attacks.

D. Denial of Service Attack

In our locating scheme, once received an alert of pollution from one of the node, the controller triggers the locating process. This involves collecting report vectors from every node in the network. An adversary can exhaust the resource of the network by flooding the controller with alerts. Our locating scheme can combat this denial of service attack in a couple of ways. Recall that the locating process is only triggered once per generation since the controller only issues one request of report per unique generation id. This already limits the effect of the attack. In addition, the controller can maintain a counter, ctr , per node. Every time a node N report $(id || N)$ and it turns out that there is no polluted edge, ctr is incremented. The controller then ignore any report from N for a period of time if ctr exceeds a certain threshold τ . After this period of time, the counter resets. Note that a malicious node N cannot pretend to be another node N' when sending an alert, *i.e.*, sending $(id || N')$ because the alert will be authenticated by the controller using the private key, $\bar{k}_{N'}^3$, shared by the controller and node N' .

E. Malicious Receivers

The assumption that the receivers are trustworthy safeguards our detection scheme against the scenarios where there are adjacent colluding adversaries. Here we discuss several available options we could adopt when we relax this assumption and consider the case where some (but not all) receivers are malicious.

One option is to use a separate key for each receiver. When some of the receivers are malicious, it is necessary for the source to share with each receiver, R_i , a separate secret key, k_i^* , instead of having all receivers and the source share a single key k^* . This is because if only one key k^* is used, a malicious receiver can leak the key to an intermediate node; as a result, this node can generate a valid end-to-end tag for any corrupted packet. For this option, for each k_i^* , the source generates a different end-to-end SpaceMac tag; thus, an honest receiver is still able to detect the pollution attack. This approach clearly increases the communication overhead of the end-to-end detection by $|\mathcal{R}|$ times as a packet now carries $|\mathcal{R}|$ end-to-end tags instead of one. This option works when the number of receivers in the network is small.

Another option is to allow alternative ways of detection. Recall that our location scheme works independently of any detection scheme used. If an honest receiver, R , receives a corrupted packet, besides relying on the end-to-end MAC tag to detect corruption, it can also use other knowledge for detection. For instance, as soon as the packets R received form an inconsistent system of equations, R knows there is an attack. Also, R can rely on application-level information to determine corrupted packets. For instance, assume R is able to solve the system but it gets corrupted packets after solving the system, and assume that this is a video packet. The corrupted packet is very likely not compliant with the expected video codec. Using this application-level information, R can detect the pollution as well. As soon as R detects the pollution attack and alerts the controller, the locating process kicks in. Recall that for any generation which experiences a pollution attack, our locating scheme can eliminate at least one attacker. A round of elimination by the locating scheme may break the adjacency property of the attackers, thus enabling the in-network detection to work in the next generation.

IX. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our detection and locating schemes. We also compare the overhead of our schemes to recently proposed schemes. In addition, we implement SpaceMac as an open-source library and we make it available online. Finally, we simulate the scenario when there are multiple adversaries and show that our locating scheme can eliminate all of them within a few generations.

A. Key Management Overhead

We compare the number of MAC keys that each verifying node needs to maintain in our defense system to that required by state-of-the-art schemes. We start by comparing the overhead of our detection scheme to those of the other MAC-based detection schemes [10], [11]. Similar to our detection scheme, the schemes in [10] and [11] also require each node to manage multiple keys. The number of keys could be large in both [10] and [11]. In particular, in [10], the number of keys each node maintains is, on average, no less than $\frac{\exp(c+1)^2 \ln(|\mathcal{V}|-1)}{c+1}$, where c is the colluding parameter. Hence, the larger c is, and/or the larger the number of nodes the network has, the more keys are needed. In RIPPLE [11], keys expire quickly periodically, and new keys are needed frequently; the number of keys increases linearly in the number of time intervals, as the transmission progresses. Clearly, the more time it takes to transmit a generation, and/or the more generations are transmitted, the larger the number of keys is needed. Standing in stark contrast to [10] and [11], the number of keys a node needs to manage in our detection scheme neither depends on the transmission time nor c : it equals to the number of the parents and children the node has (plus one). The number of children and parents of a node may or may not depend on the network size, depending on the network topology. Finally, the number of keys needed for our locating scheme is equal to that of the scheme in [20]. However, we stress that our keys can be used for multiple generations while this is not the case in [20] due to replay attacks).

Scheme	Overhead (bits)	In-network Detection	Collusion Resistance	Tag Pollution Resistance
Our scheme	$3 \lceil \log_2 q \rceil$	Yes	Yes, arbitrary resistance	Yes
RIPPLE [11]	$\frac{\ell}{2} \lceil \log_2 q \rceil$	Yes	Yes, arbitrary resistance	Yes
Broadcast MAC [10]	$ \mathbb{B} \lceil \log_2 q \rceil$	Yes	Yes, c resistance	No

TABLE II

Communication overhead of detection schemes which use homomorphic MACs along with the supported features

q	λ	δ	θ	$\Pr[P]$	$\Pr[N]$	Space Overhead
2^8	19	9	3	2^{-10}	2^{-17}	20 bytes
2^8	24	12	3	2^{-14}	2^{-16}	25 bytes
2^8	29	14	4	2^{-16}	2^{-21}	30 bytes

TABLE III

The probability a malicious parent succeeds in preventing its child to report, $\Pr[P]$, the probability a malicious child succeeds in disparaging its parent, $\Pr[N]$, and the space overhead correspond to different parameter sets.

B. Communication Overhead

Communication overhead refers to the additional network bandwidth that our schemes introduce to the system. For both of the schemes, we neglect the bandwidth of the bootstrapping phase, where symmetric keys are distributed, as this can be done offline.

Detection scheme. For the online overhead per packet, our detection scheme requires each packet to carry three SpaceMac tags: an end-to-end tag, a helper tag, and a verification tag. Each tag is a symbol in the field \mathbb{F}_q ; hence, the total overhead is $3 \lceil \log_2 q \rceil = 3 \lceil \log_2 q \rceil$ bits. Our communication overhead is fixed, regardless of the network topology.

Unlike our scheme, the online overhead of the scheme proposed by Li *et al.* [11] varies depending on the network level. In particular, in [11], the authors define a level of a node N as the length of the longest path from S to N . The network level ℓ is defined as the maximum among the levels of the nodes. In their scheme, each packet carries ℓ MAC tags initially, then one or more tags are peeled off at every node the packet goes through. The average overhead is therefore approximately $\frac{\ell}{2} \lceil \log_2 q \rceil$ bits, which is linear in the network level.

In [10], to achieve security $\frac{1}{q^d}$ and c -collusion resistance, i.e., secure against any c colluding attackers, each packet carries $|\mathbb{X}|$ MAC tags and each node verifies $|\mathbb{B}|$ tags, where (\mathbb{X}, \mathbb{B}) is a (c, d) -cover free family. For instance, to provide security $\frac{1}{q}$ and 2-collusion resistance, each packet needs to carry 49 tags and each node verifies 7 out of these 49 tags [10]. The overhead is $49 \lceil \log_2 q \rceil$ in this case, or $|\mathbb{B}| \lceil \log_2 q \rceil$ in general.

Compared to these two schemes, our detection scheme is able to provide in-network detection with significantly less communication overhead because of two main reasons: (i) we exploit local subspace property (Lemma 1) and (ii) we delegate the handling of colluding attackers to our locating scheme. Table II summarizes the overhead of our detection scheme in comparison to the other two schemes along with the supported features.

Locating scheme. The communication overhead of our locating scheme includes the λ tags carried by each packet, the reporting vectors sent by the nodes, and the announcement (containing the identified adversaries) sent by the controller. The online overhead, which depends on the number of packets

sent in the network is the first one; the latter two overhead exist only when there is a pollution attack detected, thus are asymptotically negligible in the number of packets. The overhead per packet is $\lambda \lceil \log_2 q \rceil$ bits. Table III shows that with an overhead of about 20 bytes per packet, the probability that a malicious parent succeeds in preventing its child to report, $\Pr[P]$, and the probability that a malicious child succeeds in disparaging its parent, $\Pr[N]$, are both very small.

Compared to the scheme by Wang *et al.* [20], we have the same amount of online overhead per packet (λ tags). However, in [20], when a pollution attack is detected, the controller has to compute multiple checksums for the polluted generation and send these checksums to all the nodes. Each checksum includes m symbols in F_q (recall that m is the number of packets per generation). If μ checksums are computed ($\mu > 1$ to improve the security guarantee), the overhead resulted in sending the checksums to all nodes is $|\mathcal{V}| \mu m \lceil \log_2 q \rceil$ bits. In contrast, our locating scheme does not need this checksum dissemination.

Combined Scheme. The total online communication overhead of our defense scheme is $(3 + \lambda) \lceil \log_2 q \rceil$ bits. Note that this overhead neither depends on the packet size nor the generation size; hence, it becomes more inexpensive when the packet size is large. For instance, for $\lambda = 19, n = 1024, m = 32, q = 2^8$, the per-packet communication overhead is only 2% while the security of the detection is 2^{-8} and of the security of the locating is 2^{-10} . Most importantly, standing in stark contrast to the other two schemes: [10] and [11], our overhead is constant in terms of the network size and the number of attackers.

C. Computation Overhead

The major computational overhead of both of our schemes are from the algorithms Mac, Combine, and Verify performed at each node for every packet. The computation cost for the bootstrapping, reporting, and locating steps is a one-time cost for a generation and thus is asymptotically negligible in the number of packets. We subsequently focus on the online computation cost incurred by the three algorithms of SpaceMac.

Both the Mac and the Verify algorithms incur one PRG call, m PRF calls, and $(n + 2m)$ finite field multiplications. Note that the results of both the PRG and PRF calls can be cached and used for the whole generation. Thus, they can be considered as a one-time cost as well. If we let w be the average number of packets combined by each node, then the Combine algorithm incurs w multiplications on average.

Detection Scheme. The operations performed by each node in the detection scheme for each packet y include (i) verifying the integrity of y using the Verify algorithm, (ii)

combining the received helper tags to generate a verification tag for an outgoing packet z using the Combine algorithm, (iii) computing a helper tag for z using the Mac algorithm. If the node is the source, it needs to compute the end-to-end tag using the Mac algorithm; however, in this case it does not need to verify the integrity of packets. If it is a receiver, it needs to verify the end-to-end tag by performing another Verify algorithm. The worst case computational overhead, *i.e.*, when the node is a receiver, in terms of the number of finite field multiplications is $3(n + 2m) + w$.

Locating Scheme. For each packet received, each node verifies δ tags using the Verify algorithm. For each packet it sends out, each node needs to compute λ tags using the Mac algorithm. The total overhead is therefore $(\delta + \lambda)(n + 2m)$ number of multiplications.

Combined Scheme. The overall computational overhead per packet per node of our combined scheme is $(3 + \delta + \lambda)(n + 2m) + w$.

Comparison. The computational overhead per node per packet of the scheme proposed by Li *et al.* [11] includes one Combine and one Verify operation. Based on the closed-form formulas provided in [11], this overhead is $w(\frac{\ell-1}{2}) + (n + m + \frac{\ell-1}{2})$ number of multiplications. Finally, the computational overhead per node per packet of the scheme proposed by Agrawal and Boneh [10] includes one Combine and $|\mathbb{B}|$ Verify, which is $w|\mathbb{X}| + |\mathbb{B}|(n + 2m)$ number of multiplications.

For a concrete comparison, let $q = 2^8, n = 1024, m = 32, w = 4, \ell = 9$ (for an average network of size 100, note that $\log_2 100 \approx 6$). Table IV shows the set of appropriate parameters and their corresponding computation overhead to achieve the security 2^{-8} for all schemes. Following [10], we implement multiplication in \mathbb{F}_{2^8} by creating an offline multiplication table, storing all 2^{16} products of pairs of elements in this field. The table only occupies about 64 KB in C/C++ and 128 KB in Java (since there is no 8-bit primitive data type in Java that has values from 0 to 2^8-1). This table enables us to achieve fast multiplication, which is now just a table lookup. Note that this approach is not possible (space-wise) when working with large field, for instance, any scheme that relies on public key cryptography, such as, [12]–[14], requires a large field size, *i.e.*, $\lceil \log_2 q \rceil \geq 128$. The details about the platforms we use are provided in Section IX-D. The numbers reported in Table IV are averaged over 10^6 multiplications.

As shown in Table IV, the computational latency of our detection scheme is in the same order of magnitude as the other two detection schemes. Table IV also shows that the latency of our combined detection-locating scheme is about 10 times higher than that of our detection scheme, or 30 times higher than that of RIPPLE.

This is the trade-off when one wants to locate and eliminate all attackers. If one chooses not to locate and eliminate the attackers, they may keep flooding their child nodes with corrupted packets along with their MAC tags. This not only wastes the child nodes' download bandwidth, but also exhausts their computational resources since they need to constantly run the verification algorithm on the corrupted packets. Furthermore, the attackers may only send out corrupted packets but not valid packets. This means that all packets they receive

	# Tags	Mac	Combine	Verify	Security
C/C++	1	28	0.02	28	2^{-8}
	4	112	0.08	112	2^{-32}
Java	1	61	0.09	61	2^{-8}
	4	244	0.36	244	2^{-32}
Android	1	2,273	0.66	2,273	2^{-8}
	4	9,092	2.64	9,092	2^{-32}

TABLE V
Computational time in μs of SpaceMac algorithms in \mathbb{F}_{2^8}

from their parent nodes are not used at all, which implies that the upload bandwidth of the parent nodes is also wasted. Apparently, the more parents and children the attackers have, and/or the larger the number of attackers the network has, the more resources are wasted due to the attack.

Nevertheless, we note that even though the combined detection-locating scheme has one order of magnitude larger computational delay than other stand-alone detection schemes, its delay is still very small, in the order of sub-millisecond on a PC or millisecond on a resource-constrained Android phone (Samsung Captivate). Therefore, when operating on PCs or smart phones, we strongly recommend using our full scheme. In scenarios where the network cannot afford the computation overhead of the locating scheme, each node may want to keep a threshold (per parent) of how many corrupted packets it detects from this parent so far, and refuse to receive packets from the parent after the number of corrupted packets crosses this threshold. This reduces the waste of the node's download bandwidth and CPU time.

D. Library

We implement all three algorithms of SpaceMac in both C/C++ and Java and provide them as a library. As mentioned before, we implement field multiplication using a look-up table. We implement addition as a simple XOR operation. Finally, we implement PRF and PRG using AES with CBC mode of operation. The AES implementation is provided by the standard *crypto* library [29] for Java implementation and *crypto++* open-source library [30] for C/C++ implementation. We make our SpaceMac library available online along with the source code [21].

This library is useful for those who want to adopt our SpaceMac scheme into their system, or those who want to deploy our proposed defense scheme. The C/C++ implementation is faster than the Java implementation; it is meant to be used by low-level or embedded devices, such as, network routers. The Java implementation, meanwhile, is useful for high-level application-layer programs, such as, peer-to-peer applications. Furthermore, the Java implementation is ready to be run on the current Android OS (Android 2.2 Froyo). This provides support for the rising implementation of network coding on smart phones, such as, the work in [31], [32], and [33].

Table V provides the benchmark of all three algorithms of SpaceMac. For the benchmark, we set $n = 1024, m = 32, w = 4$. Except for the Android benchmark, both the C/C++ and Java implementations were run on a PC with a quad-core 2.8 Ghz processor and 32 GB of RAM. Our Android device,

	RIPPLE [11]	Broadcast MAC [10]	Our Detection	Our Full System
Features	In-network detection Arbitrary collusion res. Tag-pollution res. No locating	In-network detection c-collusion res. No tag pollution res. No locating	In-network detection Arbitrary collusion res. Tag pollution res. No locating	In-network detection Arbitrary collusion res. Tag pollution res. Exact locating
Network Params.	$q = 2^8, n = 1024, m = 32, w = 4, \ell = 9$			
Scheme Params.	$ \mathbb{X} = 49, \mathbb{B} = 7, c = 2$			$\lambda = 19, \delta = 9, \theta = 3$
Security	2^{-8}			
# Multiplications	1,096	7,812	3,268	33,732
C/C++ (μs)	5.5	39.1	16.3	168.7
Java (μs)	6.6	46.9	19.6	202.4
Android (μs)	116.2	828.1	346.4	3,575.6

TABLE IV

Online computation overhead per packet per node in terms of the number of finite field multiplications, computing latency in C/C++, Java, and on an Android platform (Samsung Captivate)

Number of attackers	4	8	12	16	20
Average # of generations	1.92	3.01	4.16	4.77	5.64
Average delay (ms)	412	647	896	1,031	1,217

TABLE VI

The average number of generations and delay required to detect all attackers in a network with 50 intermediate nodes.

the Samsung Captivate, has a single 1 Ghz processor and 512 MB RAM. The reported values correspond to the averages taken over 10^5 runs of each algorithm. The most expensive operations of Mac and Verify algorithms are the PRF and PRG calls; however, we stress that these calls can be done offline. For completeness, the reported values include the cost of these calls.

From Table V, we can see that in order to achieve high security (2^{-32}), the computational latency of our C/C++ implementation is only in the order of hundreds of microseconds. Moreover, even on the Android resource-constrained device, the computational latency of Mac and Verify are still very small, only in the order of millisecond. Note that the Combine operation is several orders of magnitude faster than the Mac and Verify algorithms since it only involves finite field multiplications, which are quick table lookups. These results demonstrate that SpaceMac algorithms are fast and appropriate for practical use.

E. Simulation

We implement a simulation in Python that simulates a scenario where there are multiple colluding attackers in a network. We generate between a pair of source and receiver nodes a random directed acyclic graph network of 50 nodes using the *pygraph* library [34]. The ratio of edges to nodes is a random number in $[1, 5]$. All edges have a random end-to-end delay between 10 and 100 ms. In a single generation, 32 packets in $\mathbb{F}_{2^8}^{1056}$ ($q = 2^8, n = 1024, m = 32$) are generated and sent by the source node. The locating process is triggered multiple times, each time by an alert by the receiver.

The attackers in the network are chosen randomly from the population of 50 nodes in the network in a way that each attacker can still pollute the network even when the rest attackers are removed. The attackers pollute all of their outgoing edges. When requested by the controller, most of them honestly report their incoming subspaces; however, some of them, who have malicious parents, lie about their received

subspaces from those parents. This emulates the case where the attackers collude to manipulate the reports of their incoming subspaces.

We evaluate the average number of generations to locate all η attackers in the network, where η varies from 4 to 20. For each η , we perform the simulation for 100 rounds (varying the network topology, attacker location, and edge delays) to get the average value. We also evaluate the average delay it takes to identify all attackers, where the delay refers to the time between when the source starts sending and when all the attackers are located. The results shown in table VI indicate that we succeed in locating all attackers very quickly (about second for 20 attackers) and after much smaller than η generations (5.5 generations for 20 attackers).

X. CONCLUSION

In this work, we introduce a novel homomorphic MAC scheme for expanding space called SpaceMac. We propose a cooperative defense system against pollution attacks built on SpaceMac. To the best of our knowledge, our system is the first that can provide both in-network detection and exact locating of the attackers. In addition, our system is collusion resistant and tag-pollution resistant. Our evaluation results using real implementation in C/C++ and Java on multiple devices demonstrate that our defense scheme incurs both low communication and low computation overhead. We implemented SpaceMac as a ready-to-use library and make it available online.

REFERENCES

- [1] R. Ahlswede, N. Cai, S.-y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.
- [2] M. Kim, L. Lima, F. Zhao, J. Barros, M. Medard, R. Koetter, T. Kalkert, and K. J. Han, "On counteracting Byzantine attacks in network coded peer-to-peer networks," *IEEE JSAC*, vol. 28, no. 5, pp. 692–702, Jun. 2010.
- [3] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical Defenses Against Pollution Attacks in Intra-Flow Network Coding for Wireless Mesh Networks," in *ACM WiSec'09*, 2009.
- [4] C. Gkantsidis and P. R. Rodriguez, "Cooperative Security for Network Coding File Distribution," in *IEEE INFOCOM'06*, 2006.
- [5] N. Cai and R. W. Yeung, "Secure Network Coding," in *ISIT'02*, vol. 28, no. 3, Apr. 2002.
- [6] S. Jaggi, M. Langberg, T. Ho, and M. Effros, "Correction of Adversarial Errors in Networks," in *SIT'05*, 2005.
- [7] R. Koetter and F. R. Kschischang, "Coding for Errors and Erasures in Random Network Coding," in *ISIT'07*, 2007.

- [8] Z. Zhang, "Network Error Correction Coding in Packetized Networks," in *Info Theory Workshop*, no. d, 2006, pp. 433–437.
- [9] M. N. Krohn, M. J. Freedman, and D. Mazieres, "On-the-Fly Verification of Rateless Erasure Codes for Efficient Content Distribution," in *IEEE SP'04*, no. Section VIII, 2004.
- [10] S. Agrawal and D. Boneh, "Homomorphic MACs : MAC-Based Integrity for Network Coding," in *ACNS'09*, 2009.
- [11] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "RIPPLE Authentication for Network Coding," in *IEEE INFOCOM'10*, 2010.
- [12] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a Linear Subspace : Signature Schemes for Network Coding," in *PKC'09*, 2009.
- [13] D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," in *Info Sciences and Systems*, vol. 1, no. 1, 2006.
- [14] F. Zhao, T. Kalkert, M. Medard, and K. J. Han, "Signatures for Content Distribution with Network Coding," in *ISIT'07*, 2007.
- [15] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. R. Karger, "Byzantine Modification Detection in Multicast Networks using Randomized Network Coding," in *ISIT'04*, 2004.
- [16] E. Kehdi and B. Li, "Null Keys : Limiting Malicious Attacks Via Null Space Properties of Network Coding," in *IEEE INFOCOM'09*, 2009, pp. 1224–1232.
- [17] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An Efficient Scheme for Securing XOR Network Coding against Pollution Attacks," in *IEEE INFOCOM'09*, 2009, pp. 406–414.
- [18] P. Zhang, Y. Jiang, C. Lin, H. Yao, A. Wasef, and X. S. Shen, "Padding for Orthogonality : Efficient Subspace Authentication for Network Coding," in *IEEE INFOCOM'11*, 2011.
- [19] M. Jafarisiavoshani, C. Fragouli, and S. Diggavi, "On Locating Byzantine Attackers," in *NetCod'08*, 2008.
- [20] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "Identifying Malicious Nodes in Network-Coding- Based Peer-to-Peer Streaming Networks," in *IEEE Mini INFOCOM'10*, 2010.
- [21] A. Le, "SpaceMac Library," 2011. [Online]. Available: <http://www.ics.uci.edu/~anhml/software.html#SpaceMac>
- [22] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard, "Resilient Network Coding in the Presence of Byzantine Adversaries," in *IEEE INFOCOM'07*, Jun. 2007.
- [23] Q. Li, D.-M. Chiu, and J. C. Lui, "On the practical and security issues of batch content distribution via network coding," in *IEEE ICNP'06*, 2006, pp. 158–167.
- [24] A. Le and A. Markopoulou, "Locating Byzantine Attackers in Intra-Session Network Coding using SpaceMac," in *NetCod'10*, 2010.
- [25] M. Wang and B. Li, "R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *IEEE JSAC*, vol. 25, no. 9, Dec. 2007.
- [26] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman & Hall/CRC Press, 2007.
- [27] A. Khalili, J. Katz, and W. A. Arbaugh, "Toward Secure Key Distribution in Truly Ad-Hoc Networks," in *IEEE/IPSJ International Symposium on Applications and the Internet Workshops*, 2003.
- [28] M. Jafarisiavoshani, C. Fragouli, and S. Diggavi, "Subspace Properties of Randomized Network Coding," in *Info Theory Workshop*, 2007.
- [29] "Java Standard Crypto Library." [Online]. Available: <http://download.oracle.com/javase/6/docs/api/javax/crypto/package-summary.html>
- [30] "Crypto++ Library." [Online]. Available: <http://www.cryptopp.com/>
- [31] F. H. P. Fitzek, M. V. Pedersen, J. Heide, and M. Médard, "Network Coding : Applications and Implementations on Mobile Devices," in *ACM MSWiM*, 2010.
- [32] H. Shojania and B. Li, "Random Network Coding on the iPhone : Fact or Fiction ?" in *NOSSDAV*, 2009.
- [33] —, "Tenor : Making Coding Practical from Servers to Smartphones," in *ACM Multimedia*, 2010.
- [34] P. Mantiello, "Python Graph Library." [Online]. Available: <http://code.google.com/p/python-graph/>
- [4] C. Gkantsidis and P. R. Rodriguez, "Cooperative Security for Network Coding File Distribution," in *IEEE INFOCOM'06*, 2006.
- [5] N. Cai and R. W. Yeung, "Secure Network Coding," in *ISIT'02*, vol. 28, no. 3, Apr. 2002.
- [6] S. Jaggi, M. Langberg, T. Ho, and M. Effros, "Correction of Adversarial Errors in Networks," in *SIT'05*, 2005.
- [7] R. Koetter and F. R. Kschischang, "Coding for Errors and Erasures in Random Network Coding," in *ISIT'07*, 2007.
- [8] Z. Zhang, "Network Error Correction Coding in Packetized Networks," in *Info Theory Workshop*, no. d, 2006, pp. 433–437.
- [9] M. N. Krohn, M. J. Freedman, and D. Mazieres, "On-the-Fly Verification of Rateless Erasure Codes for Efficient Content Distribution," in *IEEE SP'04*, no. Section VIII, 2004.
- [10] S. Agrawal and D. Boneh, "Homomorphic MACs : MAC-Based Integrity for Network Coding," in *ACNS'09*, 2009.
- [11] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "RIPPLE Authentication for Network Coding," in *IEEE INFOCOM'10*, 2010.
- [12] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a Linear Subspace : Signature Schemes for Network Coding," in *PKC'09*, 2009.
- [13] D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," in *Info Sciences and Systems*, vol. 1, no. 1, 2006.
- [14] F. Zhao, T. Kalkert, M. Medard, and K. J. Han, "Signatures for Content Distribution with Network Coding," in *ISIT'07*, 2007.
- [15] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. R. Karger, "Byzantine Modification Detection in Multicast Networks using Randomized Network Coding," in *ISIT'04*, 2004.
- [16] E. Kehdi and B. Li, "Null Keys : Limiting Malicious Attacks Via Null Space Properties of Network Coding," in *IEEE INFOCOM'09*, 2009, pp. 1224–1232.
- [17] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An Efficient Scheme for Securing XOR Network Coding against Pollution Attacks," in *IEEE INFOCOM'09*, 2009, pp. 406–414.
- [18] P. Zhang, Y. Jiang, C. Lin, H. Yao, A. Wasef, and X. S. Shen, "Padding for Orthogonality : Efficient Subspace Authentication for Network Coding," in *IEEE INFOCOM'11*, 2011.
- [19] M. Jafarisiavoshani, C. Fragouli, and S. Diggavi, "On Locating Byzantine Attackers," in *NetCod'08*, 2008.
- [20] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "Identifying Malicious Nodes in Network-Coding- Based Peer-to-Peer Streaming Networks," in *IEEE Mini INFOCOM'10*, 2010.
- [21] A. Le, "SpaceMac Library," 2011. [Online]. Available: <http://www.ics.uci.edu/~anhml/software.html#SpaceMac>
- [22] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard, "Resilient Network Coding in the Presence of Byzantine Adversaries," in *IEEE INFOCOM'07*, Jun. 2007.
- [23] Q. Li, D.-M. Chiu, and J. C. Lui, "On the practical and security issues of batch content distribution via network coding," in *IEEE ICNP'06*, 2006, pp. 158–167.
- [24] A. Le and A. Markopoulou, "Locating Byzantine Attackers in Intra-Session Network Coding using SpaceMac," in *NetCod'10*, 2010.
- [25] M. Wang and B. Li, "R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *IEEE JSAC*, vol. 25, no. 9, Dec. 2007.
- [26] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman & Hall/CRC Press, 2007.
- [27] A. Khalili, J. Katz, and W. A. Arbaugh, "Toward Secure Key Distribution in Truly Ad-Hoc Networks," in *IEEE/IPSJ International Symposium on Applications and the Internet Workshops*, 2003.
- [28] M. Jafarisiavoshani, C. Fragouli, and S. Diggavi, "Subspace Properties of Randomized Network Coding," in *Info Theory Workshop*, 2007.
- [29] "Java Standard Crypto Library." [Online]. Available: <http://download.oracle.com/javase/6/docs/api/javax/crypto/package-summary.html>
- [30] "Crypto++ Library." [Online]. Available: <http://www.cryptopp.com/>
- [31] F. H. P. Fitzek, M. V. Pedersen, J. Heide, and M. Médard, "Network Coding : Applications and Implementations on Mobile Devices," in *ACM MSWiM*, 2010.
- [32] H. Shojania and B. Li, "Random Network Coding on the iPhone : Fact or Fiction ?" in *NOSSDAV*, 2009.
- [33] —, "Tenor : Making Coding Practical from Servers to Smartphones," in *ACM Multimedia*, 2010.
- [34] P. Mantiello, "Python Graph Library." [Online]. Available: <http://code.google.com/p/python-graph/>

REFERENCES

- [1] R. Ahlswede, N. Cai, S.-y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.
- [2] M. Kim, L. Lima, F. Zhao, J. Barros, M. Medard, R. Koetter, T. Kalkert, and K. J. Han, "On counteracting Byzantine attacks in network coded peer-to-peer networks," *IEEE JSAC*, vol. 28, no. 5, pp. 692–702, Jun. 2010.
- [3] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical Defenses Against Pollution Attacks in Intra-Flow Network Coding for Wireless Mesh Networks," in *ACM WiSec'09*, 2009.
- [4] C. Gkantsidis and P. R. Rodriguez, "Cooperative Security for Network Coding File Distribution," in *IEEE INFOCOM'06*, 2006.
- [5] N. Cai and R. W. Yeung, "Secure Network Coding," in *ISIT'02*, vol. 28, no. 3, Apr. 2002.
- [6] S. Jaggi, M. Langberg, T. Ho, and M. Effros, "Correction of Adversarial Errors in Networks," in *SIT'05*, 2005.
- [7] R. Koetter and F. R. Kschischang, "Coding for Errors and Erasures in Random Network Coding," in *ISIT'07*, 2007.
- [8] Z. Zhang, "Network Error Correction Coding in Packetized Networks," in *Info Theory Workshop*, no. d, 2006, pp. 433–437.
- [9] M. N. Krohn, M. J. Freedman, and D. Mazieres, "On-the-Fly Verification of Rateless Erasure Codes for Efficient Content Distribution," in *IEEE SP'04*, no. Section VIII, 2004.
- [10] S. Agrawal and D. Boneh, "Homomorphic MACs : MAC-Based Integrity for Network Coding," in *ACNS'09*, 2009.
- [11] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "RIPPLE Authentication for Network Coding," in *IEEE INFOCOM'10*, 2010.
- [12] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a Linear Subspace : Signature Schemes for Network Coding," in *PKC'09*, 2009.
- [13] D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," in *Info Sciences and Systems*, vol. 1, no. 1, 2006.
- [14] F. Zhao, T. Kalkert, M. Medard, and K. J. Han, "Signatures for Content Distribution with Network Coding," in *ISIT'07*, 2007.
- [15] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. R. Karger, "Byzantine Modification Detection in Multicast Networks using Randomized Network Coding," in *ISIT'04*, 2004.
- [16] E. Kehdi and B. Li, "Null Keys : Limiting Malicious Attacks Via Null Space Properties of Network Coding," in *IEEE INFOCOM'09*, 2009, pp. 1224–1232.
- [17] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An Efficient Scheme for Securing XOR Network Coding against Pollution Attacks," in *IEEE INFOCOM'09*, 2009, pp. 406–414.
- [18] P. Zhang, Y. Jiang, C. Lin, H. Yao, A. Wasef, and X. S. Shen, "Padding for Orthogonality : Efficient Subspace Authentication for Network Coding," in *IEEE INFOCOM'11*, 2011.
- [19] M. Jafarisiavoshani, C. Fragouli, and S. Diggavi, "On Locating Byzantine Attackers," in *NetCod'08*, 2008.
- [20] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "Identifying Malicious Nodes in Network-Coding- Based Peer-to-Peer Streaming Networks," in *IEEE Mini INFOCOM'10*, 2010.
- [21] A. Le, "SpaceMac Library," 2011. [Online]. Available: <http://www.ics.uci.edu/~anhml/software.html#SpaceMac>
- [22] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard, "Resilient Network Coding in the Presence of Byzantine Adversaries," in *IEEE INFOCOM'07*, Jun. 2007.
- [23] Q. Li, D.-M. Chiu, and J. C. Lui, "On the practical and security issues of batch content distribution via network coding," in *IEEE ICNP'06*, 2006, pp. 158–167.
- [24] A. Le and A. Markopoulou, "Locating Byzantine Attackers in Intra-Session Network Coding using SpaceMac," in *NetCod'10*, 2010.
- [25] M. Wang and B. Li, "R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *IEEE JSAC*, vol. 25, no. 9, Dec. 2007.
- [26] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman & Hall/CRC Press, 2007.
- [27] A. Khalili, J. Katz, and W. A. Arbaugh, "Toward Secure Key Distribution in Truly Ad-Hoc Networks," in *IEEE/IPSJ International Symposium on Applications and the Internet Workshops*, 2003.
- [28] M. Jafarisiavoshani, C. Fragouli, and S. Diggavi, "Subspace Properties of Randomized Network Coding," in *Info Theory Workshop*, 2007.
- [29] "Java Standard Crypto Library." [Online]. Available: <http://download.oracle.com/javase/6/docs/api/javax/crypto/package-summary.html>
- [30] "Crypto++ Library." [Online]. Available: <http://www.cryptopp.com/>
- [31] F. H. P. Fitzek, M. V. Pedersen, J. Heide, and M. Médard, "Network Coding : Applications and Implementations on Mobile Devices," in *ACM MSWiM*, 2010.
- [32] H. Shojania and B. Li, "Random Network Coding on the iPhone : Fact or Fiction ?" in *NOSSDAV*, 2009.
- [33] —, "Tenor : Making Coding Practical from Servers to Smartphones," in *ACM Multimedia*, 2010.
- [34] P. Mantiello, "Python Graph Library." [Online]. Available: <http://code.google.com/p/python-graph/>

APPENDIX

PROOF OF LEMMA 5

Recall that the server checks $\lambda - \delta$ tags, and in order for the server to accept a report, there must be at least θ valid tags. The probability that the child successfully forges a SpaceMac tag is $\frac{1}{q}$; and so, the probability that the child fails to forge such a tag is $1 - \frac{1}{q}$. Let i be the number of valid tags. The stated probability is a direct result of enumerating the probability of success of the child in all the cases.

PROOF OF LEMMA 6

Let \mathbf{y}_r denote the random packet of the parent's space that the child chooses to report to the controller: $\mathbf{y}_r = \sum_{j \in \mathcal{D}} \alpha_j \mathbf{y}_j$, where $\alpha_j \neq 0$ and \mathcal{D} is a subset of indices of the packets sent from the parent to the child.

Recall that the child is benign and always uses Combine to generate tag for \mathbf{y}_r . Let x denote the number of correctly computed tags of \mathbf{y}_r , *i.e.*, the parent uses Mac to compute the corresponding x tags for every \mathbf{y}_j for $j \in \mathcal{D}$. The value of x must be smaller than $\delta + \theta$ otherwise the controller will accept \mathbf{y}_r as there are at least θ valid tags. Let i out of these x tags be the number of tags verifiable by the child, $i \leq \min(\delta, x)$. Since \mathbf{y}_r has $\delta - i$ not correctly computed and verifiable by the child, there are at least $\delta - i$ not correctly computed tags which are verifiable by the child among the tags of \mathbf{y}_j 's. Thus, the probability that the child accepts all \mathbf{y}_j 's is at most $\frac{1}{q^{\delta-i}}$.

The remaining $x - i$ tags of \mathbf{y}_r are checked by the controller. These tags of \mathbf{y}_r are valid tags; hence, $x - i$ must be smaller than θ otherwise the controller will accept the report. Thus, $x - i < \theta$; hence, $i \geq \max(x - \theta + 1, 0)$. The probability that the controller rejects \mathbf{y}_r equals to the probability that there are less than θ valid tags. Since there are already $x - i$ valid tags, this probability equals

$$p_{\text{rej}} = \sum_{j=0}^{\theta-(x-i)-1} \binom{\theta-(x-i)-1}{j} \frac{1}{q^j} \left(1 - \frac{1}{q}\right)^{\lambda-\delta-(x-i)-j},$$

where j denotes the number of valid tags out of the rest $\lambda - \delta - (x - i)$ tags. Note that $p_{\text{rej}} \leq 1$.

Putting the above values together, the probability that the child accepts all \mathbf{y}_j 's which form \mathbf{y}_r and the server rejects \mathbf{y}_r when there are x correctly computed tags is upper bounded as follows:

$$\begin{aligned} p(x) &\leq \frac{1}{\binom{\lambda}{x}} \cdot \sum_{i=\max(x-\theta+1, 0)}^{\min(\delta, x)} \left[\binom{\delta}{i} \binom{\lambda-\delta}{x-i} \cdot \frac{1}{q^{\delta-i}} \cdot p_{\text{rej}} \right] \\ &\leq \frac{1}{\binom{\lambda}{x}} \cdot \sum_{i=\max(x-\theta+1, 0)}^{\min(\delta, x)} \left[\binom{\delta}{i} \binom{\lambda-\delta}{x-i} \cdot \frac{1}{q^{\delta-i}} \right] \end{aligned}$$

The best probability is the maximum of $p(x)$'s.